NISTIR 7044

Deconvolving LADAR Images of Bar Codes for Construction Site Object Recognition

[a]**Gilsinn, David E.**
[a]**Cheok, Geraldine S.**
[b]**O'Leary, Dianne P.**

[a]U. S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
  and Technology
Gaithersburg, MD 20899

[b]University of Maryland
College Park, MD 20742

NIST

**National Institute of Standards
and Technology**
Technology Administration
U.S. Department of Commerce

Deconvolving LADAR Images of Bar Codes for Construction Site Object Recognition

[a]**Gilsinn, David E.**
[a]**Cheok, Geraldine S.**
[b]**O'Leary, Dianne P.**

[a]U. S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
 and Technology
Gaithersburg, MD 20899

[b]University of Maryland
College Park, MD 20742

September 4, 2003

Deconvolving LADAR Images
Of Bar Codes for
Construction Site Object Recognition

David E. Gilsinn
Mathematical and Computational Sciences Division
Information Technology Laboratory (ITL)
National Institute of Standards and Technology
Gaithersburg, MD 20899-8910

Geraldine S. Cheok
Materials and Construction Research Division
Building and Fire Research Laboratory (BFRL)
National Institute of Standards and Technology
Gaithersburg, MD 20899- 8611

Dianne P. O'Leary
Computer Science Department
University of Maryland
College Park, MD 20742

NISTIR 7044

Certain trade names and company products are mentioned in the text or identified in an illustration in order to adequately specify the experimental procedure and equipment used. In no case does such an identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

Abstract

This report discusses a general approach to reconstructing ground truth intensity images of bar codes that have been distorted by LADAR optics. The first part of this report describes the experimental data collection of several bar code images along with experimentally obtained estimates of the LADAR beam size and configuration at various distances from the source. Mathematical models of the beam size and configuration were developed and were applied through a convolution process to a simulated set of bar code images similar to the experiment. This was done in order to estimate beam spread models (beam spread models are unique to each specific LADAR) to be used in a deconvolution process to reconstruct the original bar code images from the distorted images. In the convolution process a distorted image in vector form g is associated with a ground truth image f and each element of g is computed as a weighted average of elements of f that are neighbors to that associated element. The deconvolution process involves a least squares procedure that approximately solves a matrix equation of the form $Hf = g$ where H is a large sparse matrix that is made up of elements from the beam spread function. The results of applying the several beam spread models to deconvolving the bar code images are given. Deconvolution of data measured at 10 m was more successful than that for 20 m or 40 m. The appendices include more detailed discussion of the least squares algorithm used and sample programs used during the various phases of the analysis.

Key Words: bar codes, beam spread, deconvolution, image processing, LADAR, object recognition, sparse matrix.

Table of Contents

1.0 Introduction

Imaging sensors such as LADARs (laser distance and ranging devices) are used to rapidly acquire data of a scene to generate three dimensional (3D) models. The increased interest in this technology is due to the substantial growth in applications for real-time scene updates driven by the recent advances in imaging sensor software and hardware. Current applications include 3D modeling, site surveillance, map/terrain update/reconnaissance, bathymetry[1], indoor/outdoor visual inspection, autonomous control navigation, and collision avoidance.

Imaging sensors are used to obtain two- or three-dimensional arrays of values such as range, intensity, or other characteristics of a scene. Currently available LADARs can gather four pieces of information – range to an object, two spatial angular measurements, and the strength of the returned signal (intensity). Some instruments provide other spectral information, such as Red-Green-Blue (RGB) colors. Various methods are used to convert the data, which are collected in the form of point clouds, into meaningful 3D models of the actual environment for visualization and scene interpretation. A point cloud is a set of x, y, z points acquired by the LADAR during a scan. The need for accurate representations varies with the purpose of the application. In the construction industry, an accurate representation aids in determining payment for completed work, determining if construction errors are being made, and in tracking work progress on a project. In autonomous navigation, an accurate representation would result in crash avoidance and successful course navigation. In military target acquisition, an accurate model could mean the difference between hitting or missing a target.

The data points within the point cloud acquired by a LADAR are indistinguishable from each other with regard to their origin; i.e., there is no way to tell if a point is reflected from a tree or from a building. As a result, the methods used to generate the models treat all points identically and the results are indistinguishable "humps/bumps"on a 3D surface model of a scene. Current surface generation methods using LADAR data require intensive manual intervention to recognize, replace, and/or remove objects within a scene. This is illustrated in Fig. 1.1 where prior knowledge and human intervention was required to identify objects in the overlaid sets of data points obtained from several scans. As a result, aids to object identification have been recognized by users as a highly desirable feature and a high priority area of research.

The use of bar codes or UPC (Universal Product Code) symbols has become the universal method for the rapid identification of objects ranging from produce to airplane parts. The same method could also be used to identify objects within a construction scene. This would involve using the LADAR to "read" a bar code. The concept is to use the intensity data from the LADAR to distinguish the bar pattern. The advantage of this concept is that no additional hardware or other sensor data is required. The basis for this concept lies in the high intensity values obtained from highly reflective materials.

---

[1] Mapping of underwater terrain.
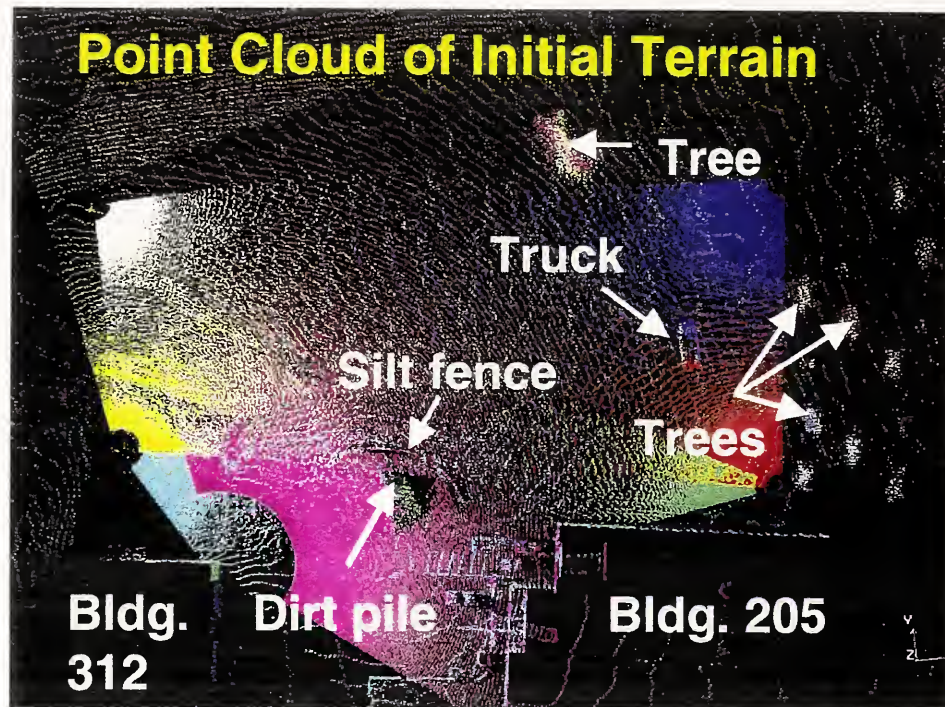
Figure 1.1. Plot of the point clouds of data acquired from several scans
of a portion of the grounds on the site of the National Institute of
Standards and Technology (NIST) in Gaithersburg, Maryland.

Using bar codes to identify objects on construction sites leads to several challenges:
- Determine the appropriate material for the bar codes – the material has to be highly reflective and durable.
- Read bar code from 100 m or greater.
  - The typical size of a construction site is generally 150 m or greater. This translates into the ability of the instrument to capture sufficient points on the bar code for correct identification. This in turn leads to the physical size of the bar codes and a hardware requirement, the scanner's angular resolution. The two factors, bar code size and scanner resolution, are related because a scanner with better resolution would require smaller bar codes.
- Distinguish bar code points from the other points in a scene.
- Read bar codes that are skewed.

Early LADAR imaging results from test bar codes showed that at distances beyond 20 m, the intensity images were too blurred to be readable and that image processing techniques were potentially necessary to reconstruct the image. The blurring or convolution of the image is a result of the low resolution (number of pixels/unit area; a consequence of the instrument's angular resolution) of the intensity images at longer distances and of distortion of the intensity image by the LADAR optics and by data

processing. As a result, an investigation of possible methods to de-blur (deconvolve) the intensity images was conducted. Deconvolution of the image involves reversing the convolution, implying that if the convolution process was known, the image may be reconstructed.

This report documents an effort to use a specific LADAR to "read" bar codes of a highly reflective material and the effort to determine a convolution-based method to reconstruct the image. This latter effort includes developing software to convolve images based on the characteristics of the LADAR, simulations to verify the software and the reconstruction of the convolved and actual LADAR images. Section 2 describes the process of data acquisition by the LADAR. Section 3 introduces image blurring fundamentals. Section 4 describes numerical image reconstruction procedures. Section 5 discusses computational results. Section 6 gives references. Section 7 provides a summary plus a brief discussion of future work. Appendix A introduces a fast matrix-vector product algorithm. Appendix B describes the LSQR algorithm of Paige and Saunders. A listing of the software is given in Appendix C.

2.0 Data Acquisition

2.1 LADAR Specifications

A Riegl scanner, Fig. 2.1, was used for all the experiments. It returns four pieces of information – range, two spatial angular measurements and intensity. The intensity is a dimensionless quantity that ranges from 0 (least reflective) to 255 (most reflective) which is based on the strength of the return signal.

The specified accuracy of the LADAR is ±2 cm in range accuracy with a maximum range of 150 m. The field-of-view is 360° in the horizontal direction and 150° in the vertical direction. The LADAR uses a pulsed (17 ns) laser with a wavelength of 903 nm. It is mounted on a pan-tilt device whose horizontal and vertical movements are controlled by two stepper motors. The angular resolution, both horizontal and vertical, is 0.045°. The manufacturer's data states that the size of the beam as it exits the LADAR is 42 mm (W) by 25 mm (H) and the beam has a 3 mrad divergence.

Elementary calculations show that the minimum vertical or horizontal distance between pixels is approximately 8 mm, 16 mm, and 31 mm for distances of 10 m, 20 m, and 40 m, respectively.
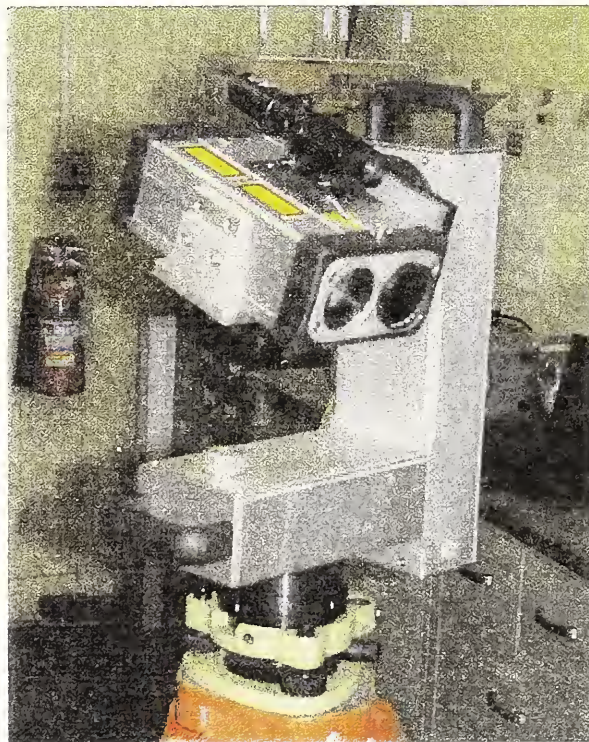
Figure 2.1. Laser Scanner.

2.2 Reflective Material

In order to "read" the bar code, its existence has to be first established. To establish its existence, the bar code has to have a unique feature or characteristic so that it is easily identifiable. Therefore, the bar code would have to be made of a material that makes it easily distinguishable from any background material based on the returned intensity value. A good candidate material would be one that that would return an intensity that was both much higher than any other material commonly found at a construction site and that was consistently high for distances of 0 m to 150 m, i.e., intensity did not drop off with distance. These requirements are essential as the returned intensity is dependent on several factors – reflectance of object, distance to object, reflectance of the surrounding objects, lighting (e.g., sunlight, shade), etc. This dependency means that the intensity of a black object at 10 m could be the same as the intensity of a shaded white object at 50 m and there would be no way to determine if the object was black or white based solely on the intensity value.

For the initial tests, 3M's Long Distance Performance (LDP) reflective sheeting was used to construct the bar codes. This material is a highly reflective prismatic lens sheeting used for traffic signage. This material was chosen as it was readily available, durable, and would reflect light even if skewed away from the light source.

The initial experiment was conducted to determine the viability of using the LDP material to fabricate the bar codes. A photograph of the LDP material and several magnified images of individual reflective prisms are shown in Figs. 2.2 and 2.3.
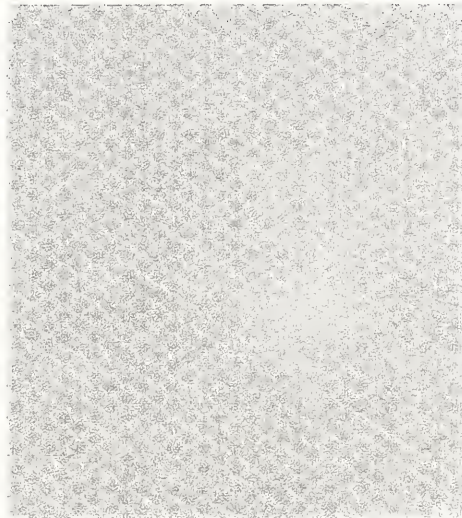
Figure 2.2. Photograph of 3M LDP Material.
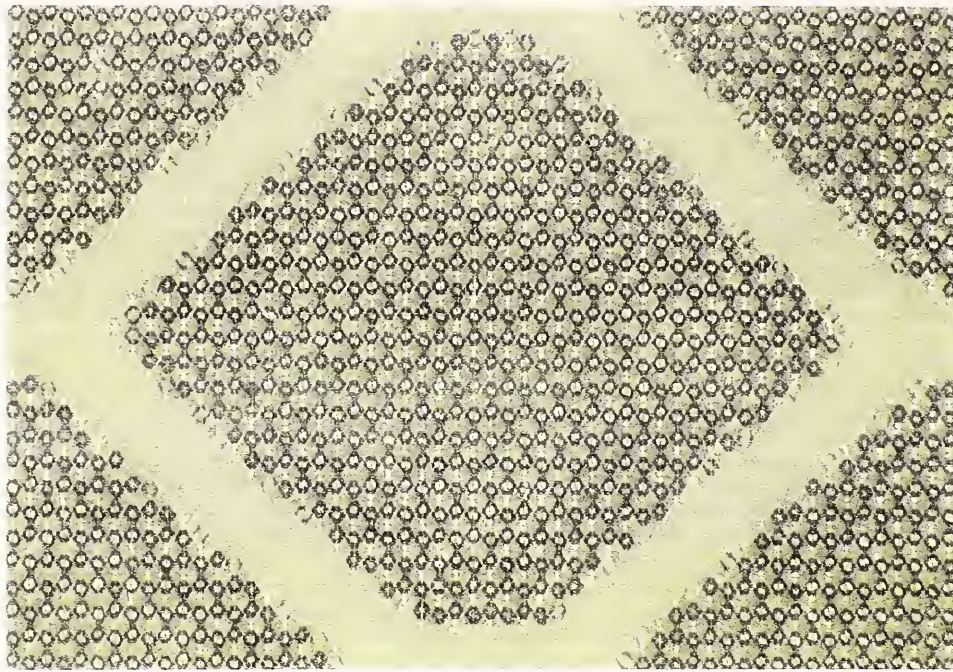
Figure 2.3a: Magnified Photo of LDP Material: Width represents 6 mm of surface.
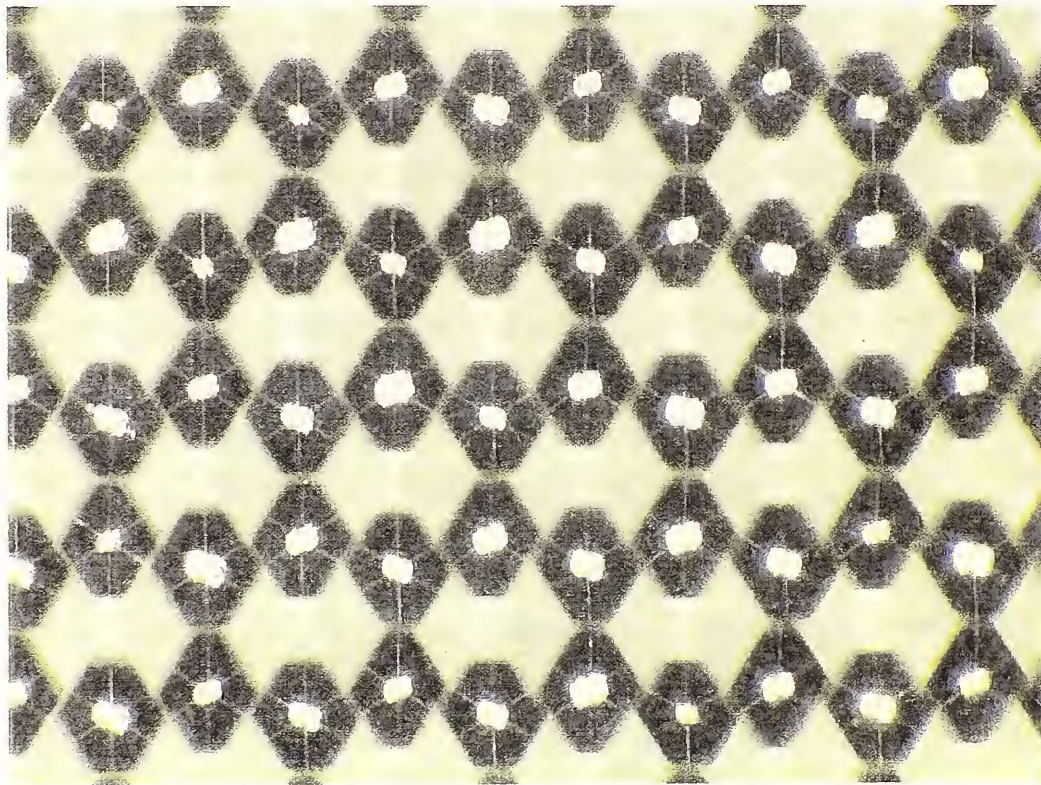
Figure 2.3b. Magnified Photo of LDP Material: Width represents 1.5 mm of surface.

2.3 Reflectivity vs Distance Experiment

The effect on the intensity value as a function of distance was examined by taking readings at 10 m intervals from 10 m to 150 m. Four targets were created using aluminum sheets, 508 mm (H) x 406 mm (W) (20 in x 16 in), that were: 1) painted matte black; 2) painted matte white; 3) left unpainted (shiny silver); and 4) covered with a sheet of LDP material. In anticipation of the need to read bar codes angled away from the scanner, the LDP target was rotated to three positions -- 0°, 45°, and 60° (Fig. 2.4).

Target rotation = 45°

Target rotation = 60°

0°

Laser
Beam

Angle
of incidence =
target rotation

Scanner

Scanner

Scanner

Top View of Set-up
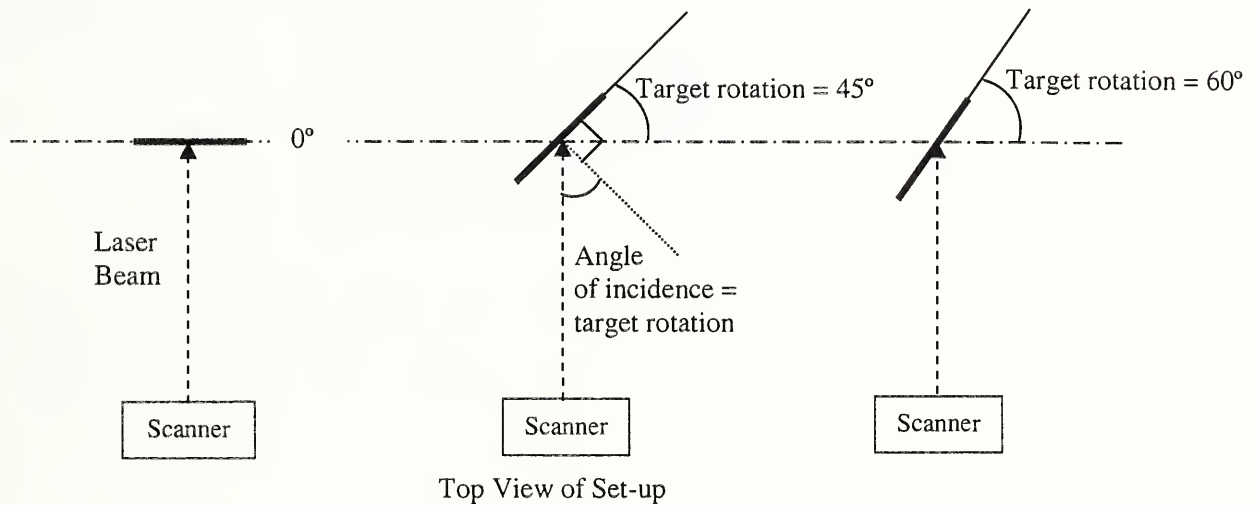
Figure 2.4. Rotation Orientation of LDP Target.

At each distance, 10 readings were recorded. The plot of the average intensity vs. distance is shown in Fig. 2.5. As seen in Fig. 2.5, the intensity values for the LDP target at 0° are consistently high -- 200 to 250 over the entire range of the scanner -- and are easily distinguishable from the other targets; thereby making these points easily distinguishable from the other points in a typical scene. As expected, the intensity values drop off when the target is turned 45° away from the scanner - the intensity values for the LDP target at 45° are very similar to those for the shiny silver target at 0° and one could be mistaken for the other. At an angle of incidence of 60°, the intensity values for a LDP target would be indistinguishable from values for the white target. However, these results are encouraging and indicate that the LDP material could potentially be used to fabricate bar codes or tags that can be read by a LADAR.

In Fig. 2.5, the intensity values for the black, white, and LDP at 60° targets increase at distances of 140 m and 150 m, which is contrary to expected. This increase may be attributed to the contribution of the white wall behind the target at the longer distances; the wall is located about 160 m from the scanner.
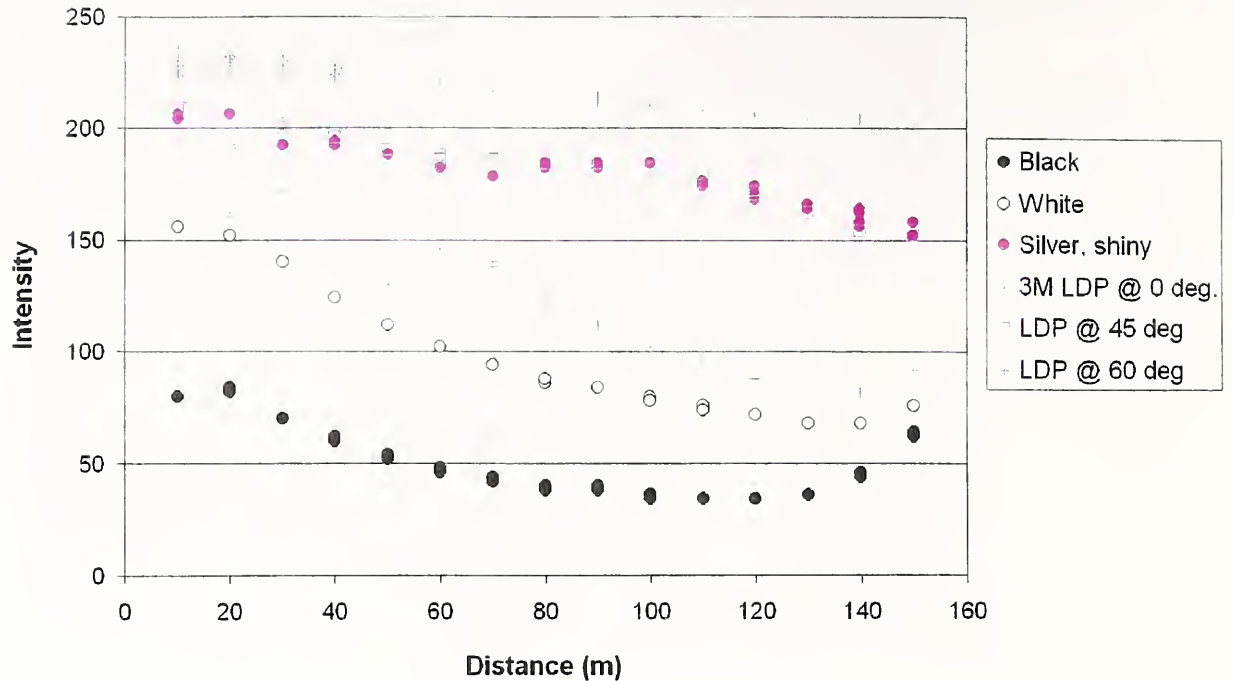
Figure 2.5. Intensity vs. Distance. The Black, White and Silver targets were at 0°.

2.4 Bar Code Patterns

Once it was determined that the LDP material produced sufficiently high intensity values and could be used to construct the bar codes, the next step was to determine if a bar code pattern could be recognized.

2.4.1 Recognizing a Simple Bar Pattern at Close Range

The first test scanned bars of varying widths, set at a fixed spacing between bars of 76.2 mm (3 in) and a distance to the target of approximately 8.7 m (28.5 ft). Three LDP bars were attached to a wooden board: 292.1 mm (H) x 152.4 mm (W) (11.5 in x 6 in), 292.1 mm x 76.2 mm (11.5 in x 3 in), and 292.1 mm x 38.1 mm (11.5 in x 1.5 in). Fig. 2.6a shows a photo of the bars and Fig. 2.6b shows a plot of the intensity values. As seen in Fig. 2.6b, the bar pattern is easily recognizable.

Figure 2.6a: Digital photograph of macro barcode test pattern.
Note that the "glare" from the bars is due to the highly reflective material of the bars.
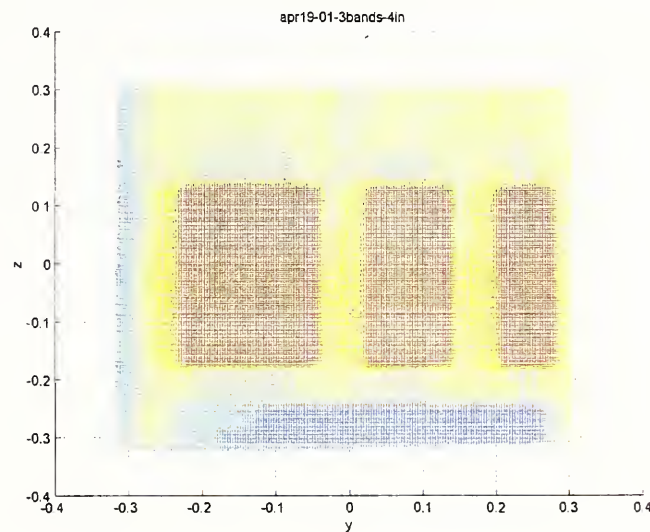
Figure 2.6b: 2-D plot of LADAR intensity at 8.7 m (28.5 ft).

2.4.2 Distinguishing Bar Patterns at Varying Distances

The next step was to test the ability to distinguish bar patterns at various distances. The objective was to determine the minimum size bar and the minimum spacing between bars required to distinguish the bar patterns at various distances. A series of test scenes were developed for various patterns and scanned at several distances.

LDP bars were attached to a white poster board 762 mm (H) x 609.6 mm (W) (30 in x 24 in). Three target boards were used. Each board contained nine bars of the same size– Board 1: 152 mm (H) x 102 mm (W) (6 in x 4 in) bars; Board 2: 152 mm x 51 mm (6 in x 2 in) bars; Board 3: 152 mm x 25 mm (6 in x 1 in) bars. The arrangement of the bars on each board was as follows:

- 3 rows of bars with 76.2 mm (3 in) spacing between the rows
- top row: 3 bars spaced at 76.2 mm (3 in)
- middle row: 3 bars spaced at 50.8 mm (2 in)
- bottom row: 3 bars spaced at 25.4 mm (1 in)

The original intent was to test the bar patterns at distances of 20 m, 40 m, 60 m, 80 m, and 100 m (65.6 ft, 131.2 ft, 196.9 ft, 262.5 ft, and 328.1 ft). The images obtained at 60 m (196.9 ft) (Fig. 2.7) showed that the LDP bars were not distinguishable at that distance. Therefore, scans were taken at shorter distances of 5 m, 10 m, 15 m, 20 m, 40 m and 60 m (16.4 ft, 32.8 ft, 49.2 ft, 65.6 ft, 131.2 ft, and 196.9 ft).

a. Intensity Image   b. Intensity Plot
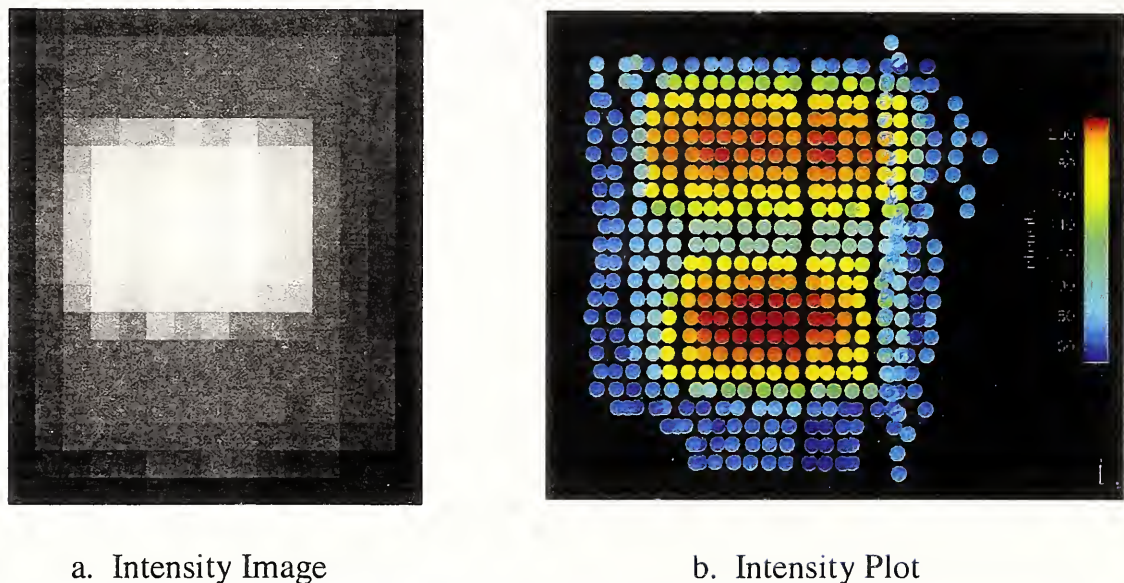
Figure 2.7. Scan of the 152.4 mm x 102 mm (6 in x 4 in) bars at 60 m (196.9 ft). Note: Middle row of bars were covered up to get better separation between the top and bottom rows of bars.

Three scans were obtained for each board at each distance. Some results are shown in Figs. 2.8, 2.9 and 2.10. In each of these figures, the (a) figure is a digital photo of the test board; figures (b), (c), and (d) represent the LADAR intensity image on a scale of 0 to 255 for various test ranges; figures (e), (f), and (g) are plots of the intensity values. In the intensity images and plots, the blurring at the bar edges is likely caused by an averaging of the intensity values when the laser beam is split between the bar and the background.
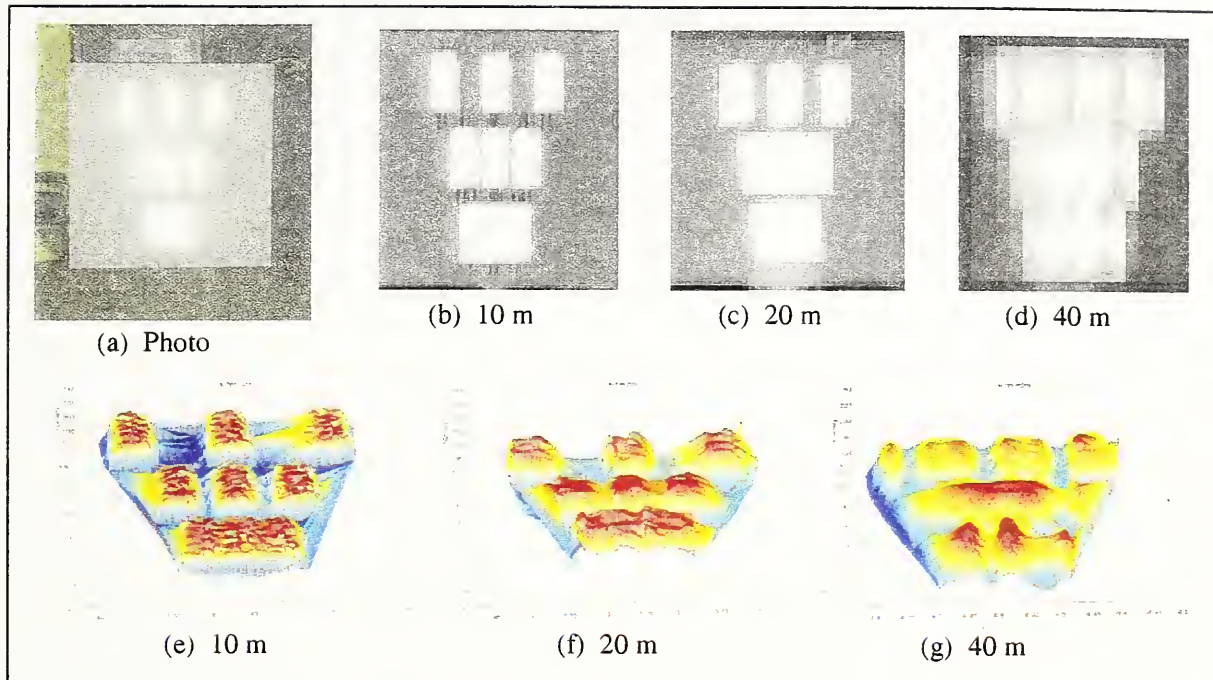
Figure 2.8. 152.4 mm x 25.4 mm (6 in x 1 in) bars at varying distances.
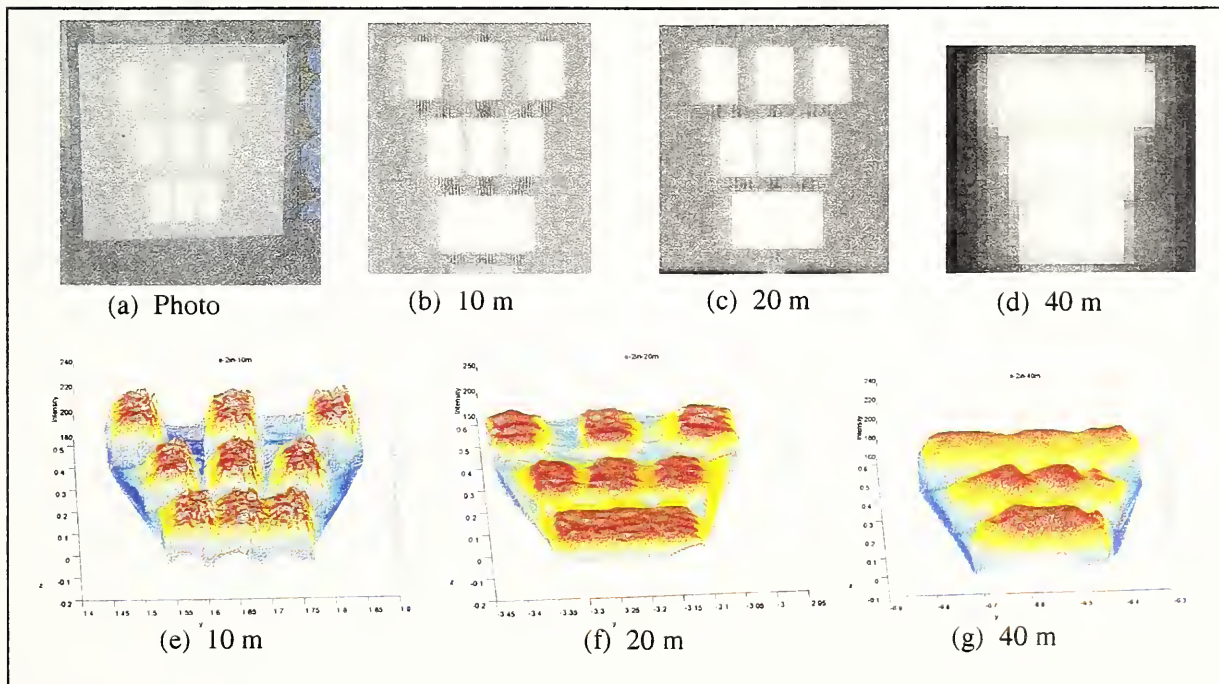
Figure 2.9: 152.4 mm x 50.8 mm (6 in x 2 in) bars at varying distances.
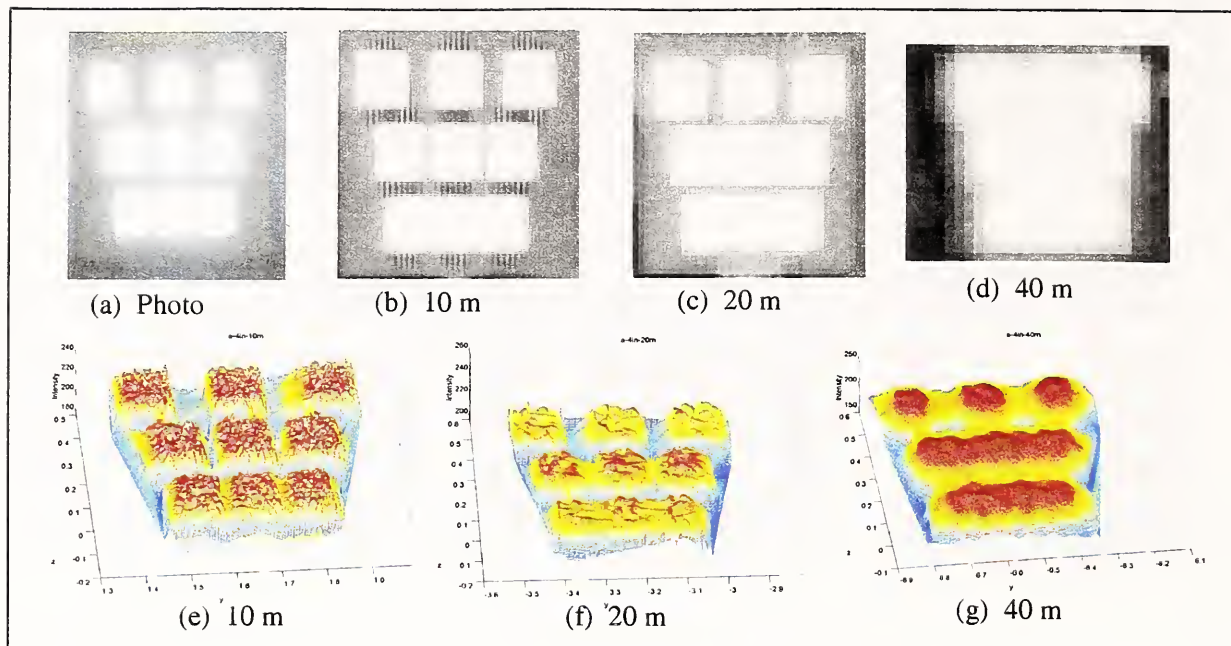
Figure 2.10: 152.4 mm x 101.6 mm (6 in x 4 in) bars at varying distances.
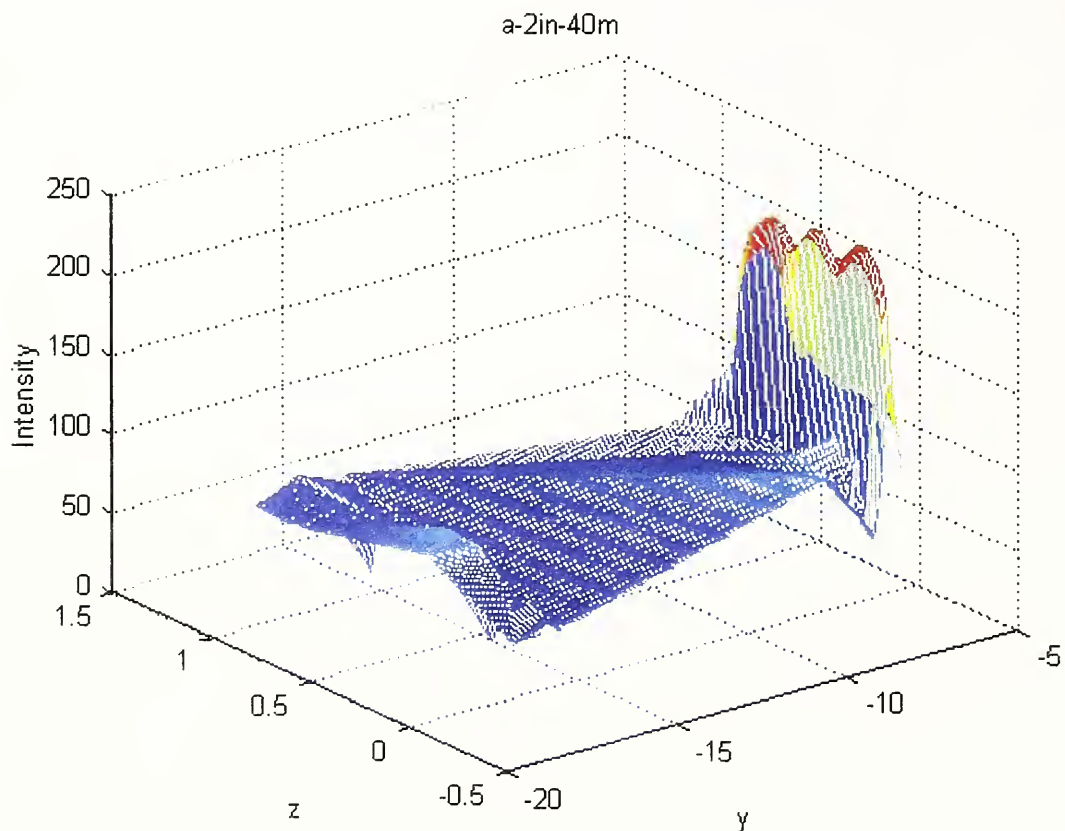
Figure 2.11: Raw data acquired by the LADAR for 50.8 mm (2 in.) bar codes at 40 m. Note the need to isolate the bar code images at the far right from the large amount of background data acquired during the scan.

In a scan, the number of points acquired can vary from several thousand to several million points. The number of data points is dependent on the desired field-of-view and desired angular resolution. Thus, the number of points returned from bar codes would be a very small percentage ($< 1\%$) of the total number of points in a typical scene. The data from one of the bar pattern experiments are shown in Fig. 2.11. The scan included the target board and a small region around the board. As seen in Fig. 2.11, the bar code points can easily be segmented or filtered out from the other background points due to the high intensity values (> 200) of the bar codes. A histogram of the intensity values, shown in Fig. 2.12, clearly shows the data segmentation. By filtering the data for points with intensities greater than 200, a cropped data set of the bar code pattern may be obtained as shown in figures (e), (f), and (g) for each of Figs. 2.8, 2.9, and 2.10. This method of was applied to all the data sets. Fig. 2.12 is a histogram of a typical raw data set.
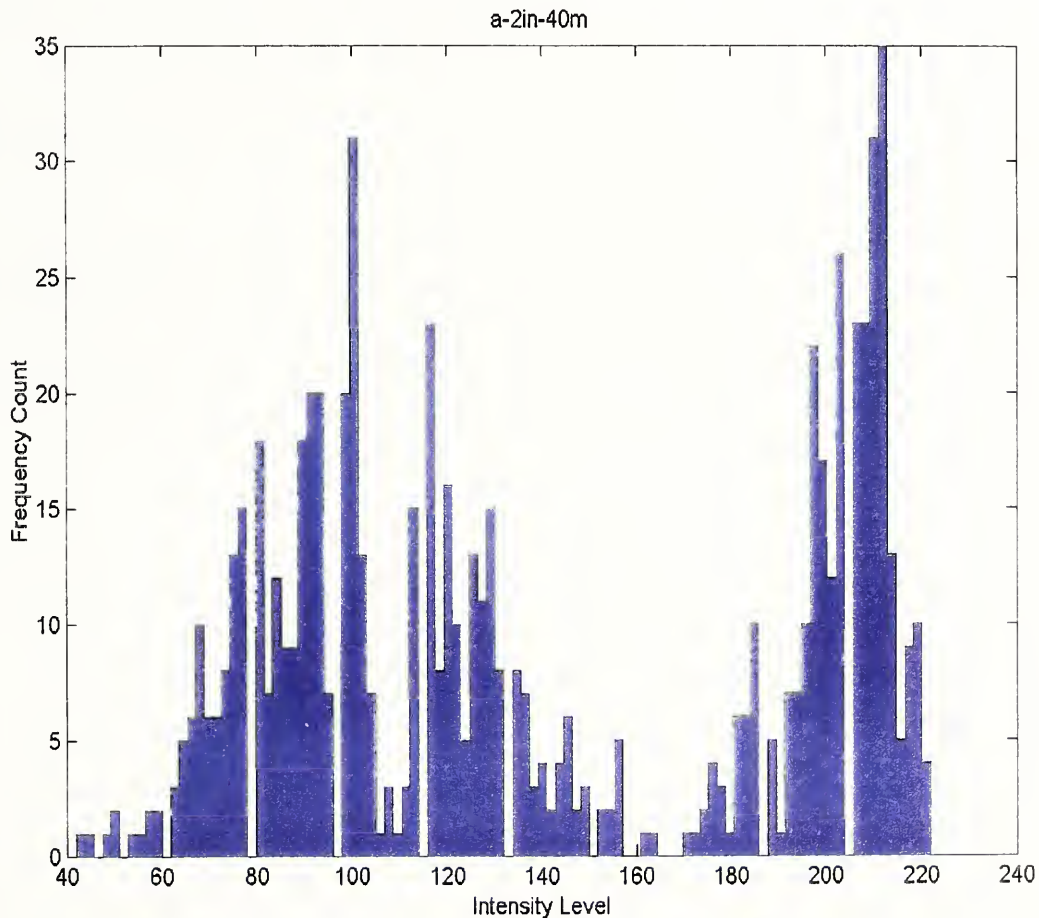
Figure 2.12. Histogram of Raw Data Set.

2.4.3 MATLAB Bar Code Identification Procedure

The MATLAB script given in Appendix C.1 employs a simple method to segment the LADAR data into background and bar data. A histogram technique is used. The returned intensity signals are binned into 100 intervals. The sample histogram shown in Fig. 2.12 displays the counts of intensities that fall within the intervals. There are two dominant modes in the distribution. The lower mode represents the large amount of background intensity shown in Fig. 2.11, whereas the higher mode represents the bar code intensities.

The script assumes the data file generated by the LADAR is in text mode with a file extension ".txt" and that there are leading header lines that begin with "#".

The use of a histogram as a filter of the intensity response offers a significant tool to isolate the intensity response from the background. The current algorithm filters out all intensity data less than 200. Further exploration of this idea will have to be based on more experimental data.

2.4.4 Bar Code Measurement Results

Two observations about the three dimensional meshing technique of the intensity data used in this paper can be made. First, the figures show that at distances of 10 m and 20 m it is possible to isolate reflector bars distanced 76.2 mm (3 in) and 50.8 mm (2 in) apart. At a separation of 25.4 mm (1 in) discrimination is problematical. A rough estimate of the separation required at 150 m (492.1 ft) for the current technology would then be 381 mm (15 in). This is based on an extrapolation of 50.8 mm (2 in) at 20 m. A further enhancement in technology is clearly required in order to reduce this interval.

The second observation is that using a histogram as a filter of the intensity response offers a significant tool to isolate the intensity response of the bar codes from the background. The utility of this tool is highly dependent on having a material with a unique intensity level. This unique intensity allows for easy data segmentation.

2.5 Beam Property Measurements

In image processing, the adequacy of the process of reconstructing an image is enhanced by understanding the entire imaging process and knowing how that process distorts the intensity measurements. Due to the proprietary nature of the particular device used for the experiments, a complete knowledge of the optics of the LADAR was nearly impossible. However, one way to estimate the dispersion of the LADAR beam and its response from an object was to measure it experimentally. In classic optics the distortion effects of an instrument can often be modeled based on measurements of bright points of light that simulate, as close as possible, a delta function. This process in classic optics is passive in the sense that the optics of the device simply measures the intensity of an external source. For LADARs, however, the process is more dynamic in the sense that the LADAR emits a beam that is reflected from an object and then the LADAR picks up the reflected beam. Thus, in the case of a LADAR there is a beam emitted, whose nature may or may not be known, there is the reflection from an object and finally there is the LADAR processing of the reflected beam. This indicates that the LADAR imaging process is a much more complex process than that of classic optics. For that reason this section describes the measurements made of a LADAR beam and the imaging of small round reflectors as an attempt to create delta functions which simulate light pulses reflected off of a point source. This was used to model the image distortion due LADAR optics.

2.5.1 Beam Size and Divergence Characteristics

The data for determining the beam size as a function of distance was obtained as part of an experiment to determine the range accuracy of the LADAR as a function of the angle of incidence of the laser beam and distance. An infrared viewer was used to see the projection of the LADAR on the target so that an outline of the beam could be drawn. Outlines of the beam were drawn by two or more observers and the measurements were averaged. The procedure of locating the beam and measuring the dimensions is detailed in another NIST report [1]. Beam dimensions were obtained for distances ranging from 2 m to 100 m and are shown in Table 2.1.

The LADAR used for this report consisted of three laser diodes and the projection of the laser beam on the target was seen as a bright rectangle for distances less than 10 m and three bright vertical bands separated by dark bands for distances greater than 10 m. The widths of the bright bands are denoted by "a", "c", and "e" and the widths of the dark bands are denoted by "b" and "d" in Table 2.1. At 10 m, one observer saw bands but the others perceived only a rectangle.

Distance (m)	L† (mm)	W† (mm)	a† (mm)	b† (mm)	c† (mm)	d† (mm)	e† (mm)
2	17	41	na	na	na	na	na
2	15	45.5	na	na	na	na	na
2	15	47	na	na	na	na	na
2	17	40	na	na	na	na	na
2	14	41	na	na	na	na	na
5	19	42	na	na	na	na	na
5	25	46	na	na	na	na	na
5	18	36	na	na	na	na	na
5	19	41	na	na	na	na	na
10[2]	31.5	49	15	2	10.5	1.5	20
10	27	51	na	na	na	na	na
10	28	48	na	na	na	na	na
10	22	46	na	na	na	na	na
10	29	58	na	na	na	na	na
10	24	53	na	na	na	na	na
10	31	62	na	na	na	na	na
20	60	68	19.5	8	16	10	14
20	53.5	68	9.5	16.5	14	14	14
20	55	56	9	15	14	11	8
30	86	79	7	21.5	11	29	10
30	79	87	9.5	24.5	15	34.5	3.5
30	74.5	86	16	18	16.5	23.5	12
39.5	117.5	116	11.5	37.5	14.5	37	15.5
39.5	96	111.5	14	35.5	13	29	20
40	115	117	17	26	20.5	31	22
40	101	105	13	24	24	30	14
50	136	159	39	32.5	14	54	19
50	127	146.5	12.5	47	22	43.5	22
60	158	166.5	21	48	23	57	17
60	161	166	24.5	53	18	47	22.5
70	180	194	20	68	25.5	62.5	18
70	166	187	16	71	18	66	16
80	176	183	14	78	18	59	14.5
80	166	207	20.5	53.5	27	84	22.5
80	172	213	19.5	97	17	57.5	22.5
90	186	260	26	115.5	25	72	21
90	254	248	22.5	73	54	61	38
90	230	230	26.5	65	44.5	64.5	29.5
100	182.5	274	25	101	34	84	30
100	234	266	17	120	14.5	101	12.5
100	246.5	263.5	15	99.5	14	122	14
100	284	288.5	21	107.5	24.5	110.5	25

[2] Although the other observers saw a single rectangular bright spot, this observer was able to distinguish bright and dark bands within the rectangular spot.

Distance (m)	L† (mm)	W† (mm)	a† (mm)	b† (mm)	c† (mm)	d† (mm)	e† (mm)
Notes:							

† Description of Variables. "a", "c", and "e" are the widths of the bright bands and "b" and "d" are the widths of the dark bands.
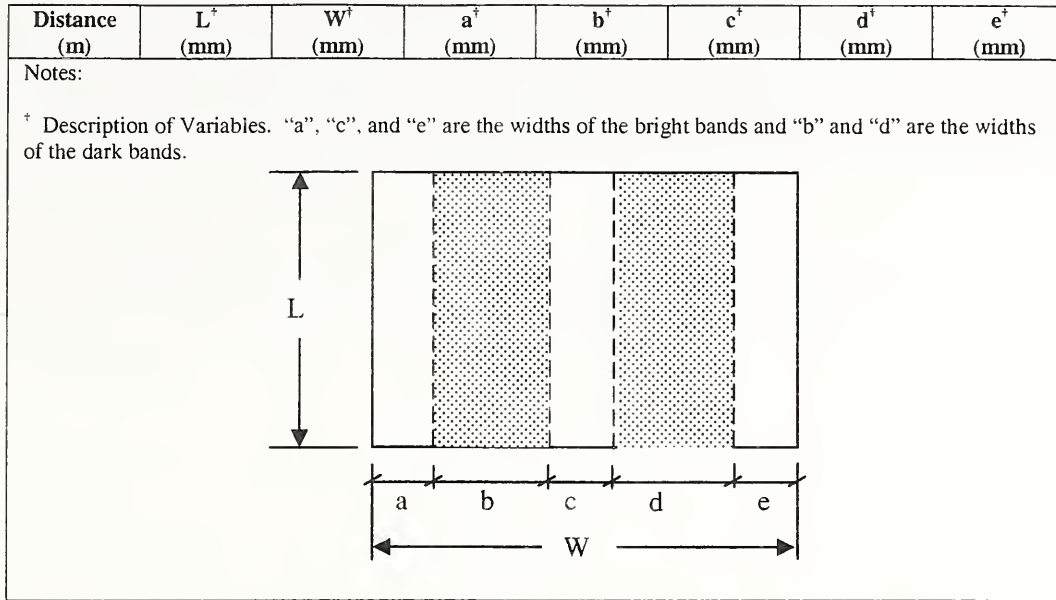
The lengths and widths of the beam projection are plotted as a function of the distance in Fig. 2.13. Given the subjectivity when obtaining the beam dimensions, a clear trend is, nevertheless, visible in Fig. 2.13. The regression fits for width and length are:

$$width = 0.007x^2 + 1.664x + 34.815 \qquad R^2 = 0.9874$$

$$length = -0.0025x^2 + 2.528x + 6.959 \qquad R^2 = 0.9606$$

where (2.1)

$$5 < x = \text{distance in meters} < 100$$

$$width, length = \text{dimension in millimeters}$$
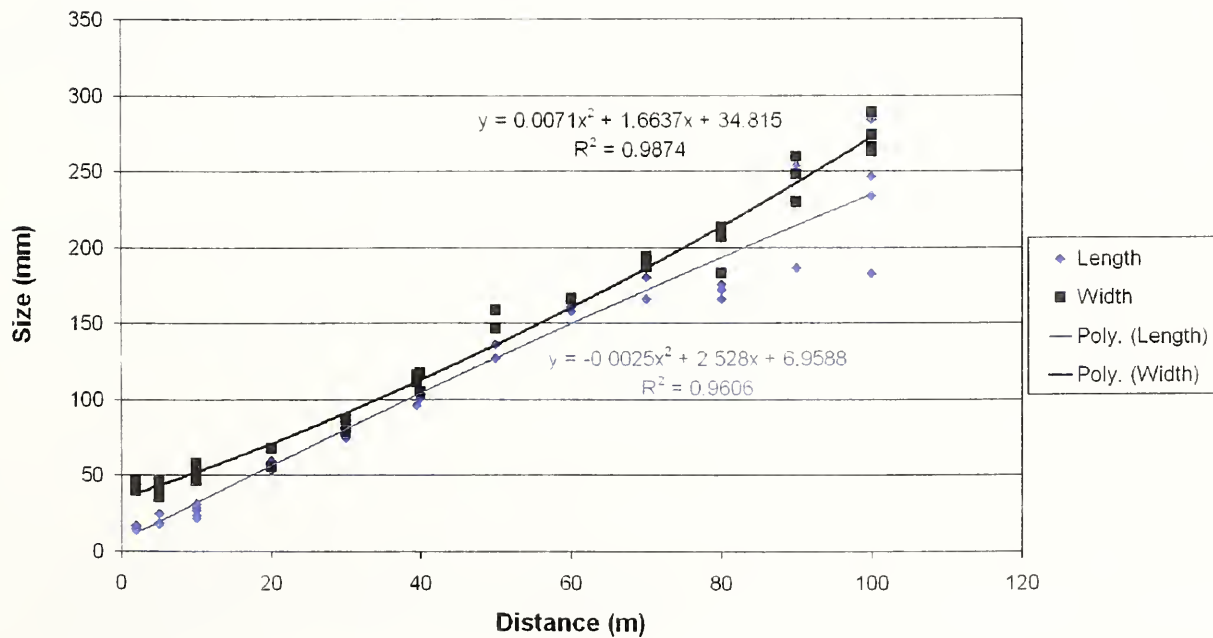
Figure 2.13: Beam Size vs. Distance.

21

Based on the measured beam dimensions, divergences of the beam in the width (horizontal) and length (vertical) directions were calculated. Since there were no measurements of the beam size as the beam exits the LADAR, the beam size at 2 m was taken as the reference or initial beam size when calculating the divergence. The divergence was calculated using the following formulas (tan $\gamma = \gamma$ for small angles):

$$\gamma_{vertical}(x) = \left[\left(\frac{L(x)-L(2)}{(x-2)\cdot 1000}\right) = \left(\frac{L(x)-15.55}{(x-2)\cdot 1000}\right)\right]\cdot 1000$$

$$\gamma_{horizontal}(x) = \left[\left(\frac{W(x)-W(2)}{(x-2)\cdot 1000}\right) = \left(\frac{W(x)-43}{(x-2)\cdot 1000}\right)\right]\cdot 1000$$

where

$\gamma_{vertical}(x), \gamma_{horizontal}(x)$ = divergence at distance x in milliradians

$L(x)$ = Average length of beam at distance x in millimeters

$W(x)$ = Average width of beam at distance x in millimeters

x = Distance in meters
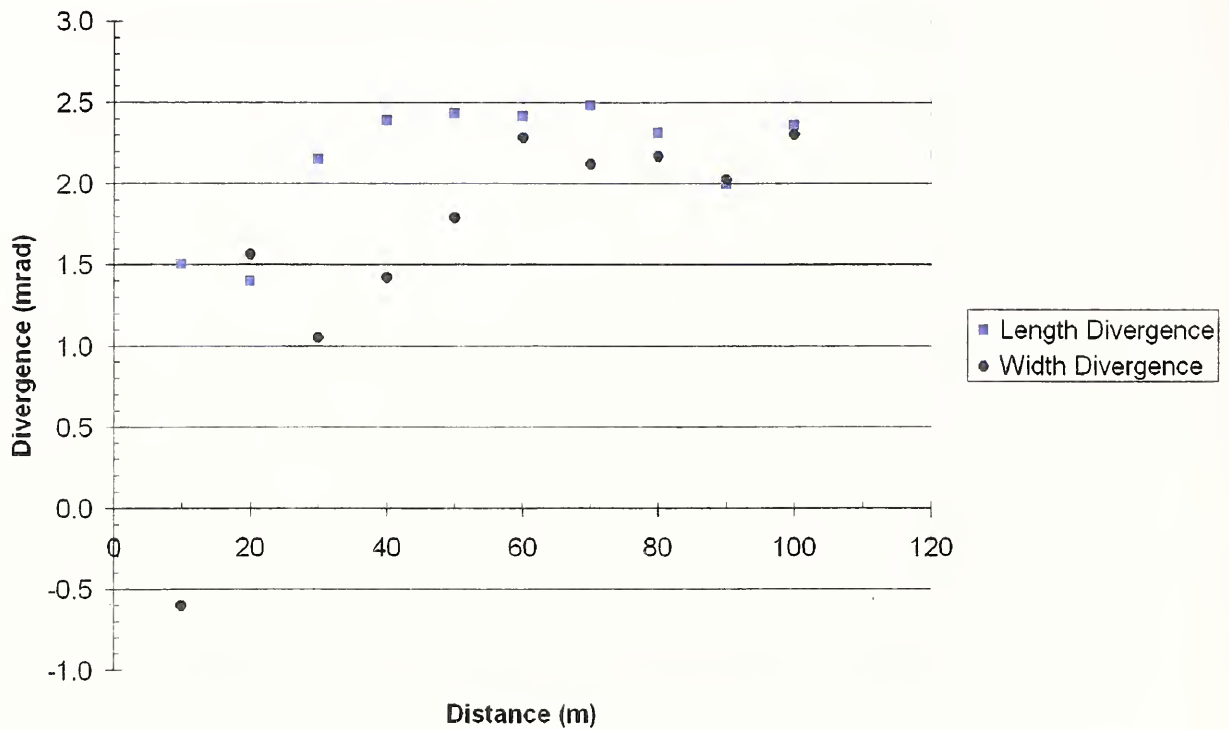
Figure 2.14. Beam Divergence.

The average vertical beam divergence is 2.14 mrad (σ = 0.39 mrad) and the average horizontal beam divergence, excluding the outlier (negative divergence) is 1.86 mrad (σ =

0.44 mrad). The average beam divergence (horizontal and vertical combined) is 2.01 mrad (σ = 0.43 mrad) - compared with the manufacturer's specified divergence of 3 mrad. The lower experimental value is likely a result of the inability of the unaided human eye to detect the faint edges of the laser beam projection. A plot of the beam divergence is shown in Fig. 2.14.

Plots of the bandwidths of the bright and dark bands are shown in Figs. 2.15 and 2.16. In Fig. 2.15, the bandwidths for the each individual band are plotted and a trend is visible for the bright and dark bands. The linear regression fits for individual bandwidths are given in Table 2.2.

Table 2.2: Coefficients for Linear Regression for Individual Bandwidths.

Description	Bandwidth[†] (mm)	Slope (M)[†]	Intercept (B)[†]	R^{2}[‡]
Bright Bands				
Left	a[§]	0.1123	11.198	0.2283
Middle	c[§]	0.1842	9.8815	0.2938
Right	e[§]	0.1315	10.643	0.2864
Dark Bands				
Left	b[§]	1.1426	-13.251	0.8954
Right	d[§]	0.9815	-6.0425	0.8702
Notes: [†] Bandwidth = Mx + B where x = distance in meters; 5 < x < 100 [‡] Correlation coefficient squared [§] Corresponds to band widths shown in Notes section of Table 2.1.				
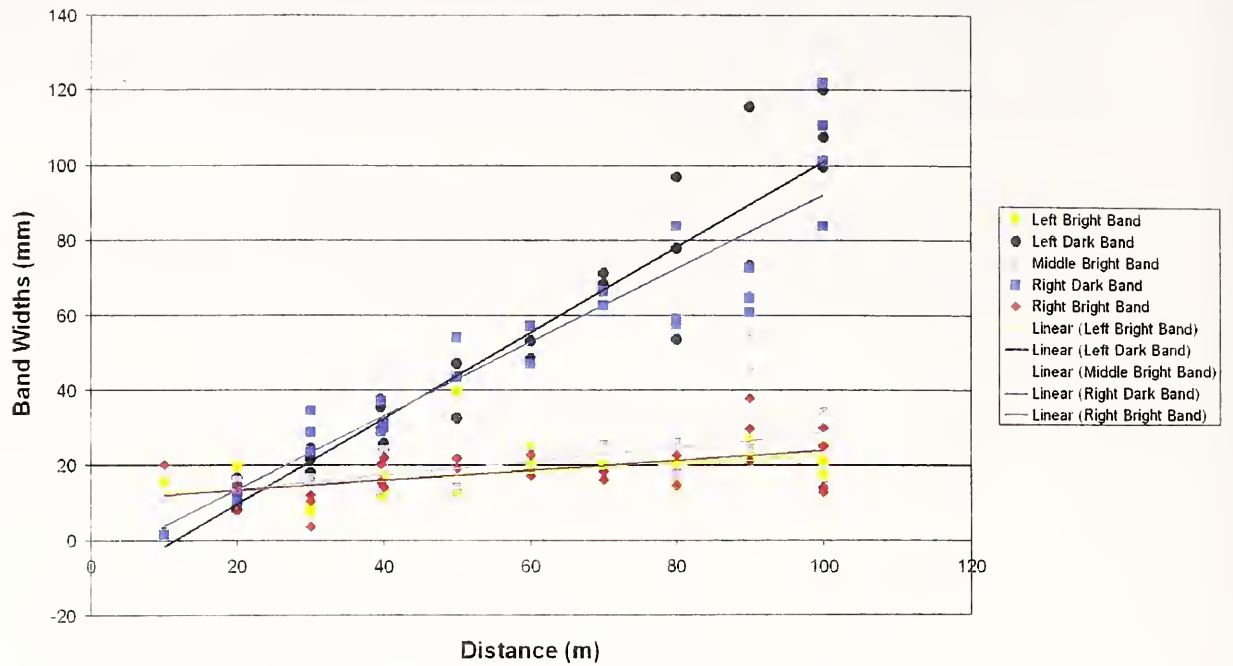
Figure 2.15. Individual Band Widths vs Distance.

Fig. 2.16 is a plot with the same data shown in to Fig. 2.15. However, in Fig. 2.16, the data for the three bright bands were combined and plotted as 'bright band' and the data for the two dark bands were combined and plotted as 'dark band'. The regression lines for the bright and dark bandwidths are:

$$\text{Dark bandwidth} = 1.062x - 9.6468 \qquad R^2 = 0.8784$$

$$\text{Bright bandwidth} = 0.1426x + 10.574 \qquad R^2 = 0.2578$$

where (2.3)

$$x = \text{distance in meters}; \quad 10 < x < 100$$

$$\text{bandwidth in millimeters}$$

24
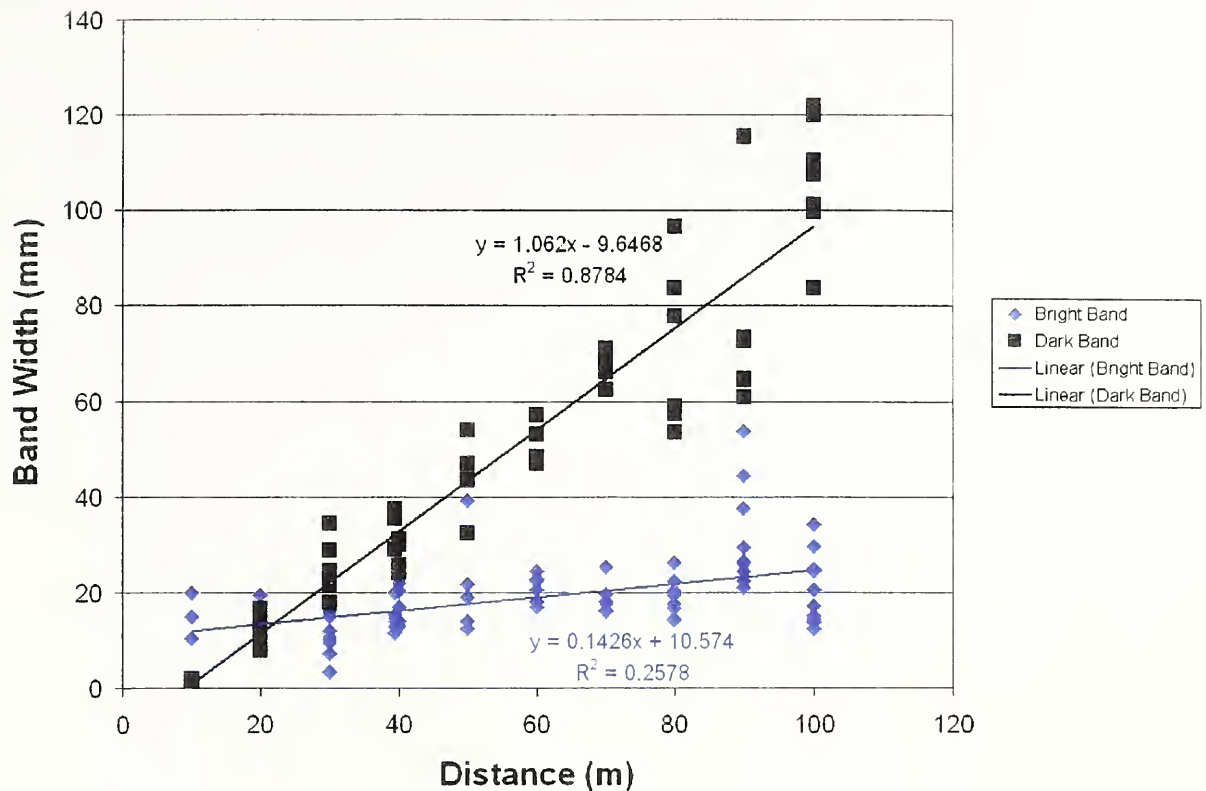
Figure 2.16. Combined Bright and Dark Band Widths vs. Distance.

2.5.2 Spread Function Measurements by Spot Reflection

In order to estimate what the beam spread response might look like, a small point of light had to be simulated. This was done by cutting two sets of circles of the 3M reflector sheet and placing them on a black background. Two diameters of circles were cut, the first being 6.3 mm (1/4 in) and the second 12.7 mm (1/2 in). Three columns of each set of "dots" were placed on two black metallic backgrounds. The dots were placed about 304.8 mm (12 in) vertically apart with the first and third colums aligned horizontally and the middle column starting 152.4 mm (6 in) below the start of the first and the third. This gave the effect of dots alternating by column every 152.4 mm (6 in).

25
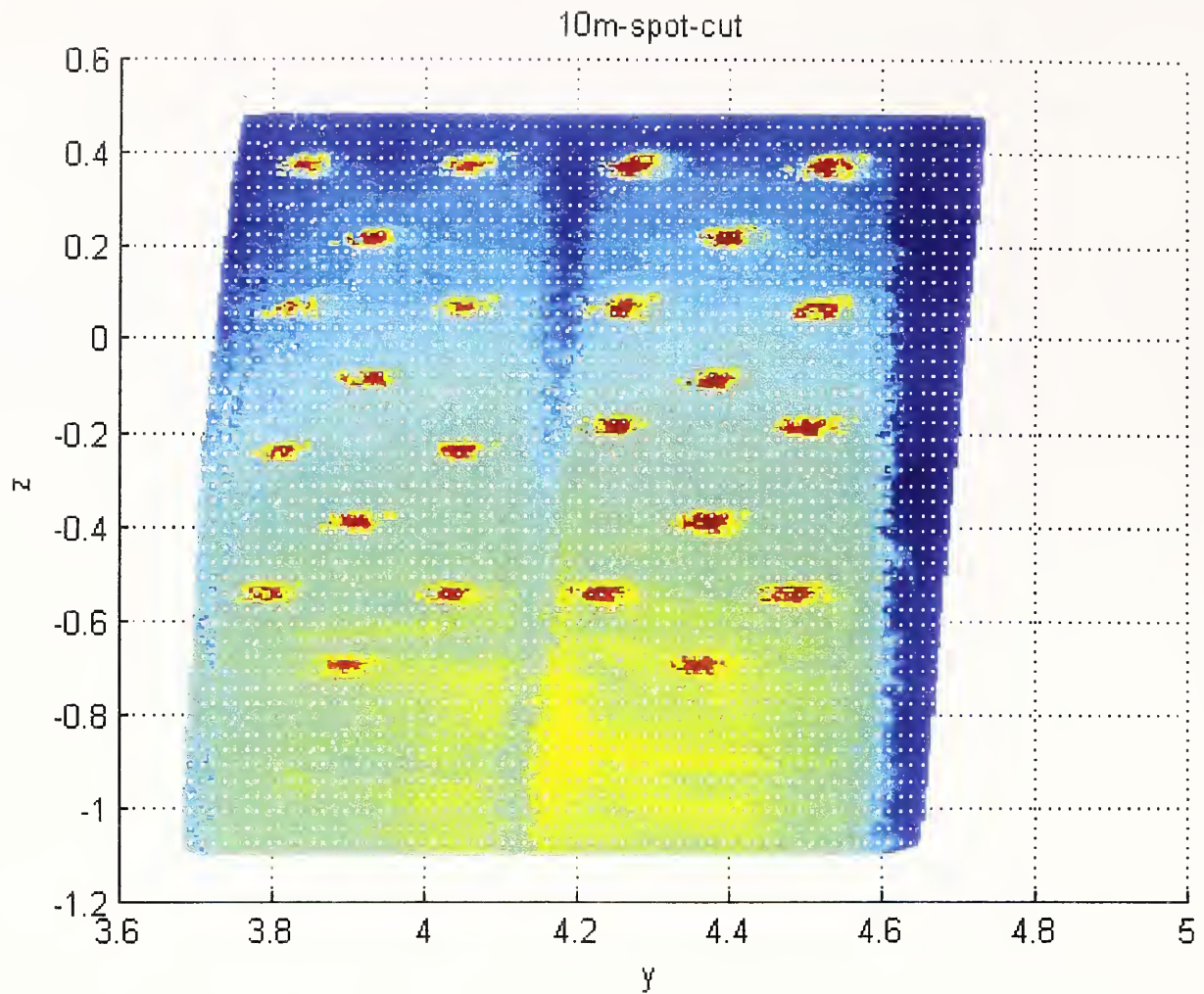
Figure 2.17: LADAR images at 10 m of 6.3 mm (1/4 in) spots in the left three columns and 12.7 mm (1/2 in) spots in the right columns.

One characterstic to notice of the spots in Fig. 2.17 is that there is a definite horizontal broadening of the spot images as opposed to a vertical broadening. In fact there is roughly a 2 to 1 aspect ratio of horizontal to vertical spread in the spot data. This aspect ratio is also evident at 20 m (Fig. 2.18). At 40 m the spots flowed together and they could not be distinguished (Fig. 2.19).

2.5.3 Summary of Beam Property Measurements

Being able to reconstruct a ground truth image from a distorted image depends strongly on knowledge of the optical processes involved. As opposed to photography, which is a passive process, in that it gathers light that is reflected from an object, a LADAR is active in that it projects a beam at a target object and then gathers in the reflected photons. Therefore, to understand a LADAR's image, one needs to have

knowledge of the beam emitted by the LADAR as well as how the LADAR optics distorts the photon beam reflected from each point of the target image.

The LADAR used in this study produced a beam of photons that split into three light bands with two dark bands between. The beam size grew nearly linearly to about 250 mm in length and width at 100 m distant. The predominant growth in the beam size was accounted for by the near linear growth in the dark bands, whereas the light band widths remained nearly constant in size.

The distortion affect of a camera's optics on light reflected from an image is usually measured by taking a picture of a small bright spot on a black background. In the case of the LADAR a bright spot was simulated by a small spot of highly reflective material against a black background. The measurements for the given LADAR showed that the distortion need not be uniform in all directions. In fact, for the LADAR used, the distortion tended to spread the image horizontally more than vertically (Fig. 2.17, 2.18) and at 40 m the reflections from the spots were so distorted that no spots in the image could be identified (Fig. 2.19).
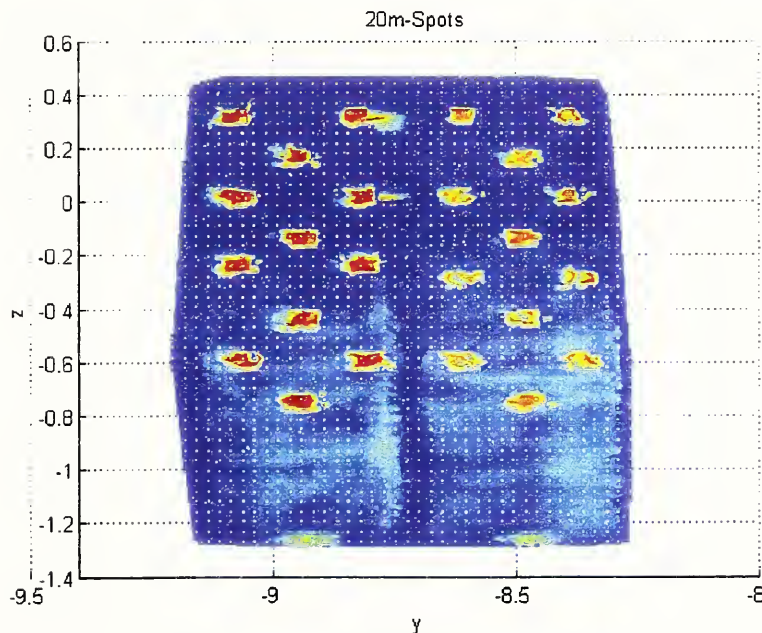
Figure 2.18. LADAR images at 20 m of 6.3 mm (1/4 in) spots in the left three columns and 12.7 mm (1/2 in) spots in the right columns.
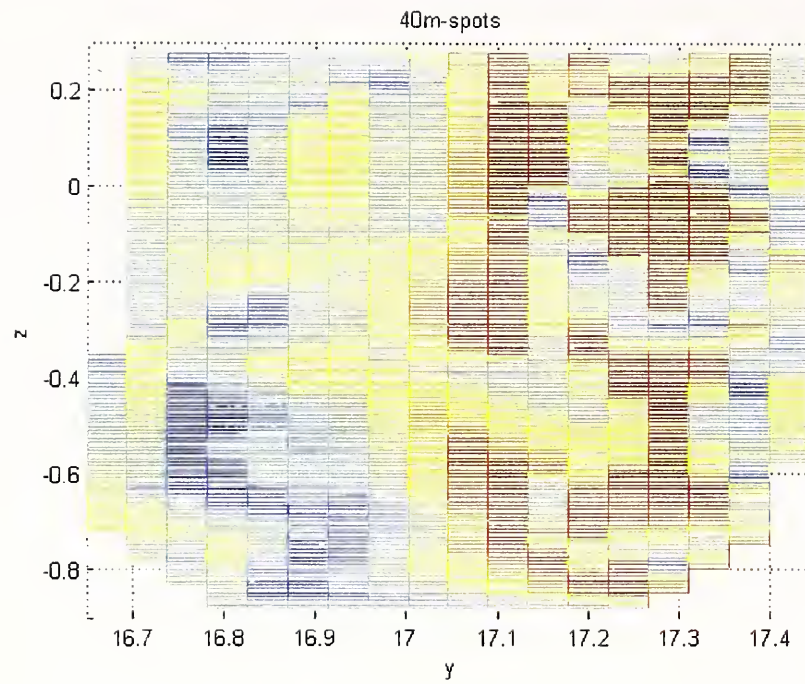
Figure 2.19. LADAR images at 40 m of 6.3 mm (1/4 in) spots in
the left three columns and 12.7 mm (1/2 in) spots in the right.

3.0 Image Blurring Fundamentals

Image processing methods have been successful when applied to blurred photographic images. They have also been successful in medical imaging. The intent of the current study was to determine to what extent image processing techniques, applied to LADAR image reconstruction problems, would be successful.

For the purpose of modeling, the LADAR beam is assumed made up of a stream of photons. The beam is aimed at a target point on the ground truth image. The value that the LADAR assigns to that point is the result of an averaging process of the reflections of the LADAR beam from points in a neighborhood of the target point as shown in Fig. 3.1. This averaging process can be given a mathematical description beginning with a discussion of a light pulse.
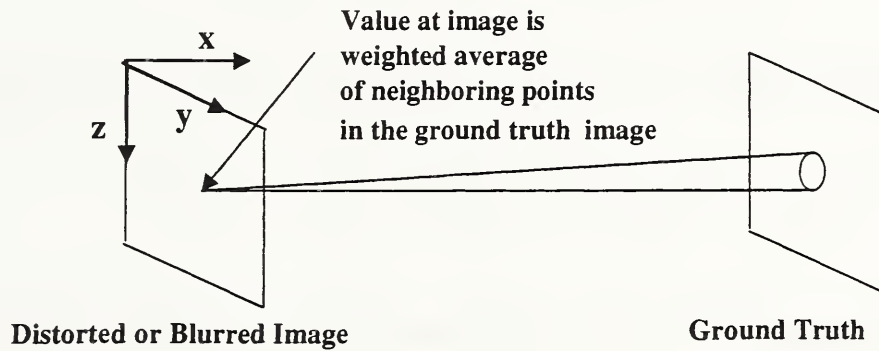
Figure 3.1. A schematic showing how an image point is assigned as a result of the blurring process that occurs after a LADAR beam is reflected.

A bright minute source of light in a dark background is essentially a highly localized, two-dimensional, spatial pulse, representing a spike of irradiance. A convenient idealized representation of this sort of sharply peaked stimulus is the Dirac delta function, $\delta(y)$. This is a quantity that is zero everywhere except at the origin, where it goes to infinity in a manner so as to encompass a unit area, that is

$$\delta(y) = \begin{cases} 0 & y \neq 0 \\ \infty & y = 0 \end{cases} \quad (3.1)$$

and

$$\int_{-\infty}^{\infty} \delta(y)dy = 1. \quad (3.2)$$

The basic operation of the delta function is called the *sifting property* and is given by

$$\int_{-\infty}^{\infty} f(y)\delta(y)dy = f(0). \quad (3.3)$$

That is the delta function extracts the one value of f(x) at x = 0. With a shift of the origin we have

$$\delta(y - y_0) = \begin{cases} 0 & y \neq y_0 \\ \infty & y = y_0 \end{cases} \quad (3.4)$$

and

$$\int_{-\infty}^{\infty} f(y)\delta(y - y_0)dy = f(y_0). \quad (3.5)$$

In two dimensions one has

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(y,z)\delta(y - y_0)\delta(z - z_0)dxdy = f(y_0, z_0). \quad (3.6)$$

The LADAR can be thought of as an optical system that takes reflected data from points on a plane, sometimes called a target plane, that are described by coordinates (y, z) where the positive y coordinate axis points to the right and the positive z coordinate axis points downward. The x coordinate is always assumed to point in the direction of the target. This axis configuration preserves the right-hand rule. The LADAR then produces an image, often blurred in some manner, on a plane, usually called the object plane, defined by coordinates (Y, Z) that are set in a one-to-one mapping with the (y, z) coordinates in the target plane.

Without knowing the physical and optical processes associated with the production of a LADAR image, a preliminary assumption to make is that it is a linear process. This can be defined as follows. Suppose that an input signal *f(y,z)* passes through some optical system and results in an output *g(Y,Z)*. The system is linear if

1. the input *af(y,z)* produces the output *ag(Y,Z)* and
2. given that input $f_1(y,z)$ produces $g_1(Y,Z)$ and $f_2(y,z)$ produces $g_2(Y,Z)$ then
   $af_1(y,z) + bf_2(y,z)$ produces $ag_1(Y,Z) + bg_2(Y,Z)$

where *a* and *b* are any scalars.

A linear system will be space invariant if changing the position of the input from the target plane merely changes the location of the output in the object plane without altering its functional form. Thus the output produced by an optical system can be treated as a linear superposition of the outputs arising from each of the individual points on the target object. The impulse response will be designated by $H(Y-y, Z-z)$. Our model of LADAR imaging can then be written as

$$g(Y,Z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(y,z)H(Y-y,Z-z)dydz. \quad (3.7)$$

The restoration problem involves estimating the function $f(y,z)$ given a measured $g(Y,Z)$. Finally there is also typically random noise degradation. In the presence of additive noise degradation the convolution restoration model can be written as

$$g(Y,Z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(y,z)H(Y-y,Z-z)dydz + n(Y,Z) \quad (3.8)$$

Equation (3.8) is a special case of a class of ill-posed problems. In order to explain the ill-posedness of the general superposition equation

$$g(Y,Z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(y,z)H(y,z;Y,Z)dydz \quad (3.9)$$

we consider two problems. The first shows that the same blurred image can be produced even though the ground truth image is perturbed by high frequency noise. The second shows that small noise perturbations of the blurred image can result from large perturbations of the ground truth image.

First consider the following one-dimensional problem. Let $f(s) = 0$ for $s \notin [-a,a]$ and then

$$g(y) = \int_{-\infty}^{\infty} f(s)H(s,y)ds = \int_{-a}^{a} f(s)H(s,y)ds \quad (3.10)$$

The Riemann-Lebesgue Theorem states that

$$\lim_{\omega \to \infty} \int_{-a}^{a} \sin(\omega s)H(s,y)ds = 0. \quad (3.11)$$

Then

$$g(y) = \lim_{\omega \to \infty} \int_{-a}^{a} [f(s) + \sin(\omega s)]H(s,y)ds. \quad (3.12)$$

The restoration problem can be stated as given $g(y)$ and $H(s, y)$ determine $f(s)$. However, since a high-frequency sinusoid can be added to $f(s)$ and produce data very close to $g(y)$, a unique solution cannot in general be expected. Other techniques have to employed.

For the second problem we begin by defining

$$k(y) = \begin{cases} 1 & |y| \leq 1/2 \\ 0 & \text{otherwise} \end{cases}. \qquad (3.13)$$

Then define

$$\delta_n(y) = nk(ny) \qquad (3.14)$$

and note that

$$\int_{-\infty}^{\infty} \delta_n(y)dy = 1. \qquad (3.15)$$

Let $H(y)$ be a bounded continuous function defined on the real line. Then for any integer $n > 0$ we can use the Mean Value Theorem for integrals to show

$$\int_{-\infty}^{\infty} \delta_n(y)H(Y-y)dy = n\int_{-1/(2n)}^{1/(2n)} H(Y-y)dy = n\int_{Y-1/(2n)}^{Y+1/(2n)} H(u)du = H(\xi) \qquad (3.16)$$

for some ξ, $Y - 1/(2n) < \xi < Y + 1/(2n)$. Now let

$$g(Y) = \int_{-\infty}^{\infty} f(y)H(Y-y)dx. \qquad (3.17)$$

Then

$$\int_{-\infty}^{\infty} \{f(y) + \varepsilon\delta_n(y)\}H(Y-y)dy = \int_{-\infty}^{\infty} f(y)H(Y-y)dy + \int_{-\infty}^{\infty} \varepsilon\delta_n(y)H(Y-y)dy \qquad (3.18)\backslash$$
$$= g(Y) + \varepsilon H(\xi)$$

for some ξ, $Y - 1/(2n) < \xi < Y + 1/(2n)$. For small ε the right hand side of (3.18) is a small perturbation of the blurred image $g(Y)$ since $H(y)$ is bounded. But the perturbation of $f(y)$ can be made arbitrarily large by selecting a sufficiently large n.

Both of these examples show how ill conditioned the image reconstruction problem can become. For this reason care must be taken when developing an image reconstruction algorithm.

Before an algorithm can be described we need to discretize the associated integrals. In general Equation (3.7) is never directly evaluated since the intensity values $f(y,z)$ are possibly known only at discrete points. To discretize the ground truth or target image define nf discrete values on the y-axis directed to the right and nf equally spaced discrete values on the z directed vertically downward so that

$$
\begin{aligned}
&0 = y_1 < y_2 < \cdots < y_{nf} = 1, \\
&0 = z_1 < z_2 < \cdots < z_{nf} = 1, \\
&\Delta y_i = y_{i+1} - y_i, \Delta z_i = z_{i+1} - z_i, \\
&\Delta y_i = \Delta z_i = \Delta.
\end{aligned}
\qquad (3.19)
$$

The unit size is set to one in order to represent a one meter square background to the bar codes used in the experiments. The intensity at the patch identified by the coordinates (y_i, z_j) is modeled as

$$
f(y_i^*, z_j^*)\Delta y_i \Delta z_j, \quad (3.20)
$$

where f is a function expressing the intensity response at some point in the patch. Due to distortions, the LADAR image of the response from the bar code surface is smeared out into some form of blurred spot.

The grid for the distorted image is taken as a subset (to be defined below) of the grid for the ground truth image for the purpose of discretization; see Fig. 3.2. Points in the distorted image will be identified by (Y, Z) and those in the ground truth image by (y, z). These are simply different notations for points in the same axis system.

The distortion at a point (Y, Z) in the object plane, due to a point (y,z) in the target plane, is described by the function $H(y,z;Y,Z)$, called here the <u>Beam Spread Function</u>, although in standard image processing it would be called a point spread function. For most practical purposes the Beam Spread Function can be considered spatially invariant in the sense that its distortion value only depends on the distance between (Y, Z) and (y, z) so that H has the form $H(Y-y, Z-z)$ as given in Equation (3.7). The incremental distortion effect at (Y, Z) due to a neighboring patch of (y,z) is then

$$
\Delta g(Y,Z) = f(y,z)H(Y-y, Z-z)\Delta y \Delta z. \quad (3.21)
$$

To describe the total effect $g(Y, Z)$ of all of the points (y, z) in the ground truth image one sums over all of the patches in the ground truth image

$$g(Y,Z) = \sum_{i=1}^{nf-1} \sum_{j=1}^{nf-1} f(y_i^*, z_j^*) H(Y - y_i^*, Z - z_j^*) \Delta y_i \Delta z_j. \quad (3.22)$$

This is the discrete form of Equation (3.7), because as the number of grid points nf in the ground truth image grows and the patch size tends to 0, the sum can be replaced by the integral (3.7). This integral is called a <u>convolution integral</u>.
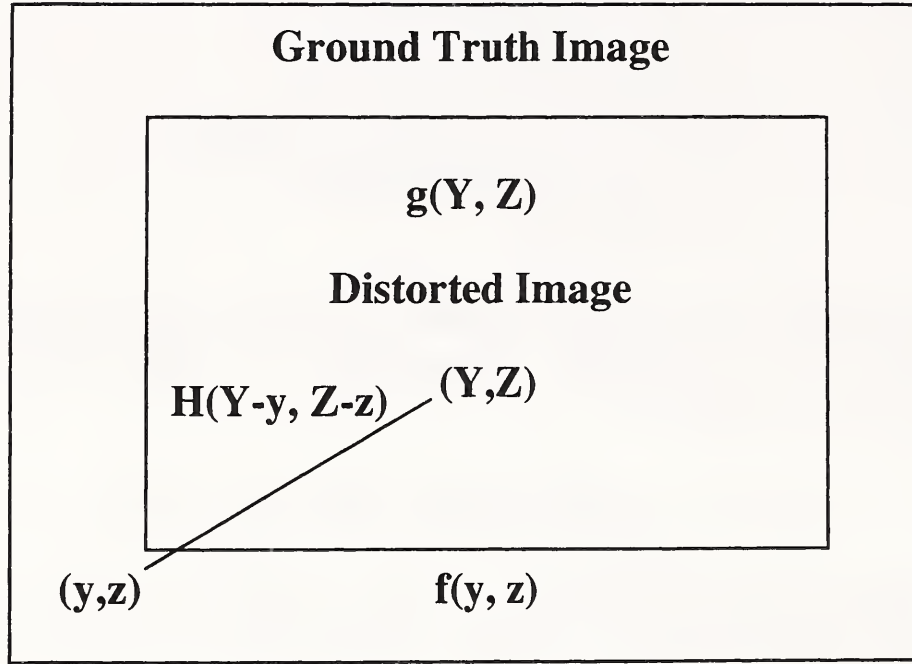
Figure 3.2. This shows the relation of the distorted image to the ground truth image and the fact that the Beam Spread Function only depends on relative distances between. points.

4.0 Numerical Image Reconstruction Procedures

In order to be in a form suitable for computer processing, an image function $f(y, z)$ must be digitized both spatially and in amplitude. Digitization of the spatial coordinates (y, z) will be referred to as *image sampling*, while amplitude digitization will be called *gray-level quantization.*

Suppose that a continuous image $f(y, z)$ is approximated by equally-spaced samples arranged in the form of an $nf \times nf$ array as shown by

$$f(y,z) = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,nf) \\ f(2,1) & f(2,2) & \cdots & f(2,nf) \\ \vdots & \vdots & \vdots & \vdots \\ f(nf,1) & f(nf,2) & \cdots & f(nf,nf) \end{bmatrix} \quad (4.1)$$

where each element of the array is a discrete quantity. The right side of this equation represents what is commonly called a *digital image*, while each element of the array is referred to as an *image element*, *picture element*, *pixel*, or *pel.*

The digitization process requires that a decision be made on a value for nf as well as on the number of discrete gray levels allowed for each pixel. It is assumed here that m discrete levels are equally spaced between 0 and 255. The *resolution* (i.e. the degree of discernable detail) of an image is strongly dependent on both nf and m. The more these parameters are increased, the closer the digitized array will approximate the original image. But computer storage and processing requirements increase rapidly as a function of nf and m. When an image of lower resolution, such as 16 x 16, is displayed at a higher resolution, of say 512 x 512, pixels are each duplicated leading to a display that has a checkerboard-like effect in the graphics display.

Assume that the ground truth image $f(y_i, z_j)$ is given as a matrix $f(i, j)$ of size $nf \times nf$ at points

$$0 = y_1 < y_2 < \cdots < y_{nf} = 1,$$
$$0 = z_1 < z_2 < \cdots < z_{nf} = 1 \quad (4.2)$$

and the beam spread function is given as a matrix $H(m,n)$ of size $ma \times ma$, $ma < nf$, where $m = 1, \ldots, ma$, $n = 1, \ldots, ma$, with $H(1, 1)$ the upper left corner and $H(ma, ma)$ the lower right. H has the same grid size as f.

The convolution is a process in which beam spread function, as a matrix of weights, is sequentially moved across the ground truth matrix. A weighted average of the ground truth points under the beam spread matrix is computed and the averaged value is assigned

to a point under the center of the beam matrix. The distorted image is a function $g(Y_p, Z_q)$ represented by a matrix $g(p,q)$ of size $ng \times ng$ where nf and ng are related by $nf = ng + ma - 1$; see Fig. 4.1. Note that ma must be taken as an odd integer.
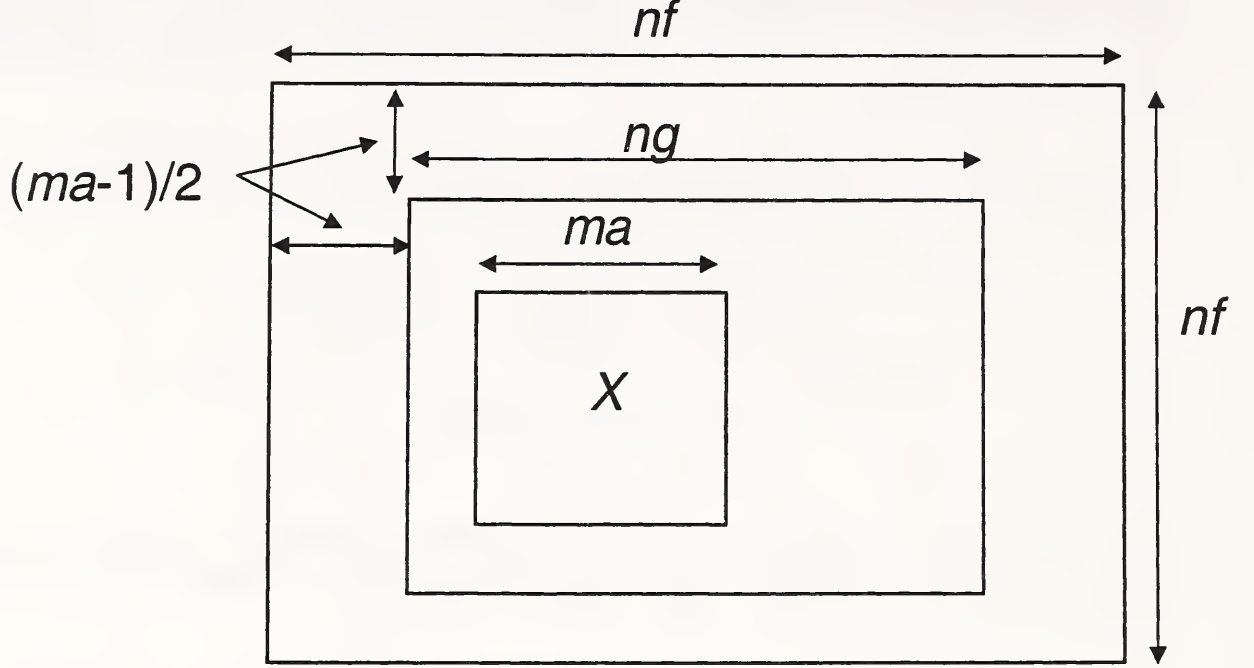
Figure 4.1. This shows the relationship of the Ground Truth image of size $nf \times nf$ to the distorted image of size $ng \times ng$, where $nf = ng + ma - 1$, where the discrete model of the LADAR beam has size $ma \times ma$. The point X represents the pixel in the distorted image at which the convolution of the beam with the Ground Truth is assigned.

The spatial coordinates for the distorted image are defined as follows. Let $p, q = 1, ...,$ ng then set

$$Y_p = (p-1)\Delta,$$
$$Z_q = (q-1)\Delta. \qquad (4.3)$$

Similarly, the spatial coordinates of the ground truth image are defined as follows. Let i, $j = 1, ..., nf$ then set

$$y_i = (i-1)\Delta,$$
$$z_j = (j-1)\Delta. \qquad (4.4)$$

Now we can define the indexed array for the beam spread function in terms of special coordinates. Although H will be used to designate an indexed array, it will also be used as the related function of coordinates. For $m, n = 1, ..., ma$ let

$$H(m,n) = H_L(u_m, w_n) \qquad (4.5)$$

where H_L designates the value of the matrix H at the local coordinates

$$u_m = ((2m - ma - 1)/2)\Delta,$$
$$w_n = ((2n - ma - 1)/2)\Delta. \qquad (4.6)$$

u_m and w_n are local coordinates of H relative to the center of the beam spread function. For example,

$$H(1,1) = H_L((-(ma-1)/2)\Delta, (-(ma-1)/2)\Delta),$$
$$H((ma+1)/2, (ma+1)/2) = H_L(0,0), \qquad (4.7)$$
$$H(ma, ma) = H_L(((ma-1)/2)\Delta, ((ma-1)/2)\Delta).$$

A point (Y_p, Z_q) in the distorted image is linked to a point (y_i, z_j) by

$$i = p + (ma-1)/2,$$
$$j = q + (ma-1)/2. \qquad (4.8)$$

For example,

$$Y_p = y_i = y_{p+(ma-1)/2} = (p + (ma-1)/2 - 1)\Delta. \qquad (4.9)$$

Therefore, given any p, i, q, j,

$$Y_p - y_i = (p - i + (ma-1)/2)\Delta,$$
$$Z_q - z_j = (q - j + (ma-1)/2)\Delta. \qquad (4.10)$$

Since $H(u,w)$ is nonzero for (u,w) in $[-((ma-1)/2)\Delta, ((ma-1)/2)\Delta] \times [-((ma-1)/2)\Delta, ((ma-1)/2)\Delta]$ and 0 elsewhere, the discrete convolution integral becomes

$$g(Y_p, Z_q) = \sum_{j=q}^{q+ma-1} \sum_{i=p}^{p+ma-1} H(Y_p - y_i, Z_q - z_j) f(y_i, z_j). \qquad (4.11)$$

Summation is first taken down y then z. The Δ factor has been subsumed into the definition of H for computational ease. Note that, if the beam spread matrix is sitting with its center at (Y_p, Z_q), the upper left point is sitting at (y_p, z_q). At point (Y_p, Z_q) of the distorted image, the sum looks like

$$g(p,q) = H(ma,ma)f(p,q) + H(ma-1,ma)f(p+1,q) + \cdots + H(1,ma)f(p+ma-1,q)$$
$$+ H(ma,ma-1)f(p,q+1) + \cdots + H(1,ma-1)f(p+ma-1,ma-1) \qquad (4.12)$$
$$+ \cdots + H(1,1)f(p+ma-1,q+ma-1).$$

Since the ground truth image F is larger than the distorted image G, there are more degrees-of-freedom involved in reconstructing F from a measured G. A computable approach is to determine F in a least squares manner to satisfy

$$\min_{F} \|HF - G\|^2. \qquad (4.13)$$

This is an ill-posed problem since there can be multiple solutions. A penalty term can be added to this minimization problem that puts a premium on the size of F selected. Introduce $\lambda > 0$ and form the following minimization problem

$$\min_{F} \left\{ \|HF - G\|^2 + \lambda \|F\|^2 \right\}. \qquad (4.14)$$

The second term is called a <u>regularization term</u> and its function is to control the magnitude of the final F. In practice λ is selected as a small positive number.

At this point we discuss an iterative least squares algorithm by Paige and Saunders [8] called LSQR. Although this has been discussed in [7] many of the details involved with the implementation in code may not be familiar to some readers of the current report. Therefore the details are included in APPENDIX B. Only the general outline will be given here. A Fortran 77 code is available in zip compressed form on the web as ACM Algorithm 583 from <u>http://www.acm.org/calgo/contents/</u>.

One begins with a fundamental result from Golub and Kahan [5]. Let H be any m x n matrix with real elements. The original result allowed complex elements but only real values will be used here. The matrix H can be decomposed as

$$H = U B V^T \qquad (4.15)$$

where U and V are orthogonal matrices and the first column of U is arbitrary. That is, $U^T U = UU^T = I$, $V^T V = VV^T = I$. B is a bidiagonal matrix of the form

$$
B = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ \beta_2 & \alpha_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \beta_n & \alpha_n \\ 0 & \cdots & 0 & 0 \\ \vdots & \cdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{bmatrix}.
\qquad (4.16)
$$

Using the orthogonality property of U and V it is clear that

$$
\begin{aligned}
HV &= UB, \\
H^T U &= VB^T,
\end{aligned}
\qquad (4.17)
$$

where

$$
B^T = \begin{bmatrix} \alpha_1 & \beta_2 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \ddots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \beta_n & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \alpha_n & 0 & \cdots & 0 \end{bmatrix}
\qquad (4.18)
$$

The previous decompositions will become useful in the solution of the following least squares problem

$$
\min_f \|Hf - g\|^2.
\qquad (4.19)
$$

If orthogonal matrices U and V are found that satisfy the above conditions then this least squares problem can be reduced to a simpler form. By orthogonality we always have $\|z\| = \|U^T z\|$ so that

$$
\|Hf - g\| = \|UBV^T f - g\| = \|U^T(UBV^T f - g)\| = \|BV^T f - U^T g\|.
\qquad (4.20)
$$

Let the columns of U be denoted by

$$
U = [u_1, \ u_2, \ \cdots, \ u_m].
\qquad (4.21)
$$

Since the first column of U is arbitray, we can set

$$
u_1 = \frac{g}{\|g\|}.
\qquad (4.22)
$$

$$U^T g \begin{bmatrix} u_1^T g \\ u_2^T g \\ \vdots \\ u_m^T g \end{bmatrix} = \begin{bmatrix} \beta_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \beta_1 \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \beta_1 e_1 \qquad (4.23)$$

where, for notation, e_k is the vector of all zeroes with one in the kth element and $\beta_1 = \|g\|$. With this notation one has

$$\|Hf - g\| = \|By - \beta_1 e_1\| \qquad (4.24)$$

where $y = V^T f$. Thus the previous least squares problem can be reduced to the new minimization problem

$$\min_y \|By - \beta_1 e_1\|. \qquad (4.25)$$

The regularization term can be introduced by solving the minimization problem

$$\min_y \left\| \begin{bmatrix} B \\ \lambda I \end{bmatrix} y - \begin{bmatrix} \beta_1 e_1 \\ 0 \end{bmatrix} \right\|. \qquad (4.26)$$

The algorithm of Paige and Saunders [7] proceeds iteratively so that after $k+1$ steps one has generated

$$\beta_1 = \|g\|,$$
$$U_{k+1} = [u_1, u_2, \cdots, u_{k+1}],$$
$$V_{k+1} = [v_1, v_2, \cdots, v_{k+1}],$$
$$B_k = \begin{bmatrix} \alpha_1 & & & & \\ \beta_2 & \alpha_2 & & & \\ & \beta_3 & \alpha_3 & & \\ & & \ddots & \ddots & \\ & & & \ddots & \ddots \\ & & & & \ddots & \alpha_k \\ & & & & & \beta_{k+1} \end{bmatrix} \qquad (4.27)$$

40

The kth approximation to the solution f is defined by $f_k = V_k y_k$ where y_k solves the kth iteration problem

$$\min_{y_k} \left\| \begin{bmatrix} B_k \\ \lambda I \end{bmatrix} y_k - \begin{bmatrix} \beta_1 e_1 \\ 0 \end{bmatrix} \right\|. \qquad (4.28)$$

If we define the following residuals

$$t_{k+1} = \beta_1 e - B_k y_k$$
$$r_k = g - H f_k \qquad (4.29)$$

then Paige and Saunders [7] show that the relations

$$r_k = U_{k+1} t_{k+1}$$
$$H^T r_k = \lambda^2 f_k + \alpha_{k+1} \tau_{k+1} v_{k+1} \qquad (4.30)$$

hold to machine accuracy. In the second equation τ_{k+1} represents the last component of $t_{k+1} = [\tau_1, \tau_2, \cdots, \tau_{k+1}]^T$. They also show that (r_k, f_k) are acceptable solutions of

$$\min_f \left\| \begin{bmatrix} H \\ \lambda I \end{bmatrix} f - \begin{bmatrix} g \\ 0 \end{bmatrix} \right\| \qquad (4.31)$$

if the values of $\|t_{k+1}\|$ or $|\alpha_{k+1} \tau_{k+1}|$ are sufficiently small.

5.0 Computational Results

Before deconvolution was applied to the original measured bar code images, a simulation was developed in order to better understand the data distortion brought about by the convolution process. These computations required several stages. First, the targets or ground truth images had to be created. There were three of those. Second, the individual beam matrices had to be generated for simulated probing of the target at 10 m, 20 m and 40 m. Next the target images, originally defined as matrices, had to be converted to vector form and the beam matrices had to be put into a form to use as convolution kernels. Next the target images were blurred by computing the convolutions. Finally a least squares deconvolution process was applied to the blurred images to retrieve the original images. The first two subsections describe creating the simulated ground truth data and the simulated blurred bar code images. The third subsection describes the experience with deconvolving the measured bar code images using various Beam Spread Functions.

5.1 Creating Simulated Ground Truth Bar Code Data

Determining the ground truth version of an image obtained by a LADAR is a nontrivial task. One solution is to declare, in the case of the bar code images, that an image acquired at say 5 m would be considered ground truth. Another might be to take a photograph of the bar code display and scale, if possible, the photograph's spatial and intensity values to approximately those of a LADAR image taken at the same distance. However, in all of these cases, the data would be affected by the blurring nature of the data acquisition device. One could agree that at certain distances these effects would be considered minimal. Another approach, however, and the one taken in this report, is to build simulated bar code data sets using the distribution of the intensities from acquired LADAR data. This approach allows one to create properly sized ground truth data sets, depending on the values of the mesh size of the acquired data (ng) and the matrix size of the beam spread function being evaluated (ma), since the number of grid points in the ground truth data set is related to these two quantities.

A MATLAB script for the simulated ground truth bars is given in Appendix C2. It generates three sets of simulated bar codes depending on bar widths. The specifications for these bar code configurations are based upon the experimental bar code artifacts. Each set of bar codes consists of three rows of bars, each of height 0.1524 m (6 in). The rows are separated by 0.0762 m (3 in). The axes are taken so that the Z-axis is directed downwards, the Y-axis toward the right and the X-axis pointed forward. The origin of the axis system is taken as the center of the middle bar of the middle row. The rows are numbered 1 to 3 starting with the lower row. The individual bars are numbered left to right beginning with the lower row as 1, 2, 3. The second row, left to right, is 4, 5, 6. Finally the upper row, left to right is 7, 8, 9. The distances between bars in the lower first row is 0.0254 m (1 in), between bars in the second row is 0.0508 m (2 in), and finally in

the third row the distance is 0.1016 m (4 in). Each of the three sets of bar codes has the same height. The three widths are 0.1016 m (4 in), 0.0508 (2 in) and 0.0254 m (1 in).

Figures 5.1 – 5.3 show the three simulated ground truth data sets.
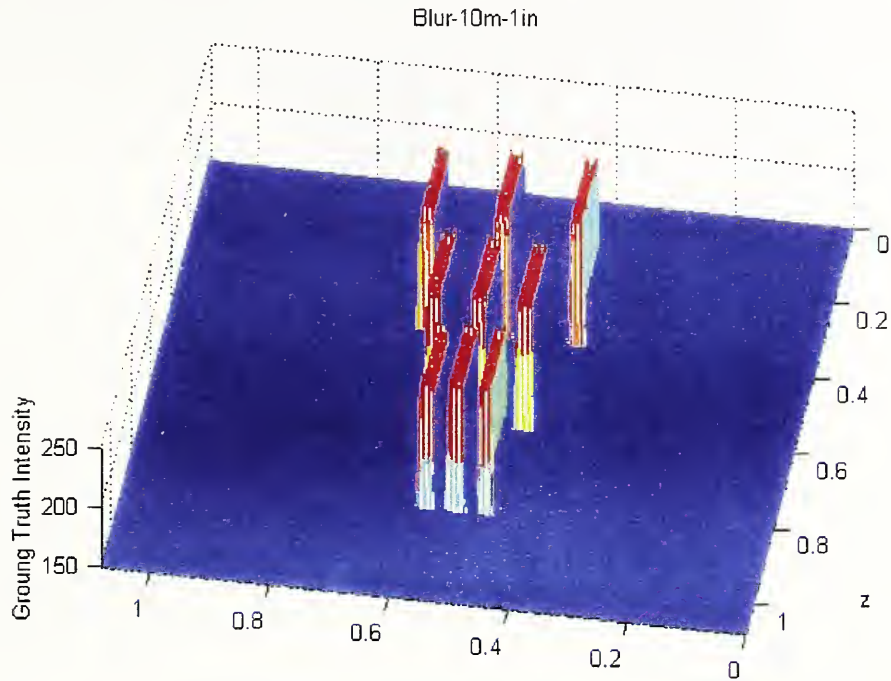
Figure 5.1. Simulated 0.0254 m (1 in) Bar Codes.
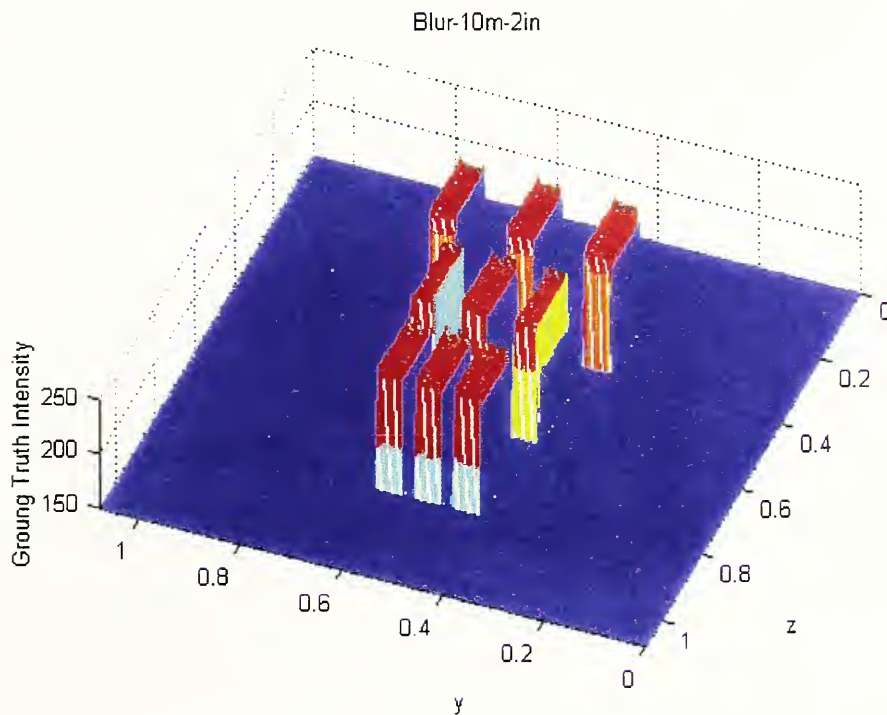
Figure 5.2. Simulated 0.0508 m (2 in) Bar Codes.
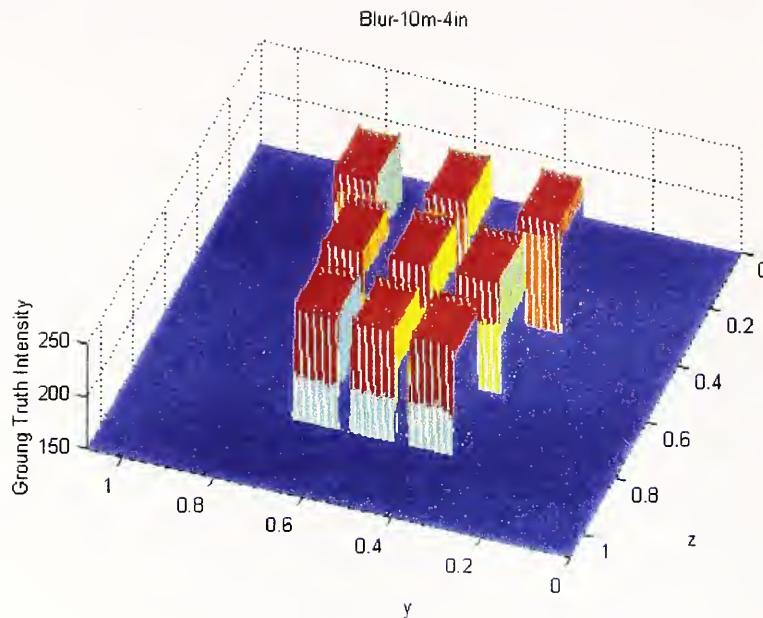
Figure 5.3. Simulated 0.1016 m (4 in) Bar Codes.

Fig. 5.4 shows the simulated 0.1016 m (4 in) bar configuration with Gaussian noise added. The colors in all of the figures are set by the default MATLAB color table.
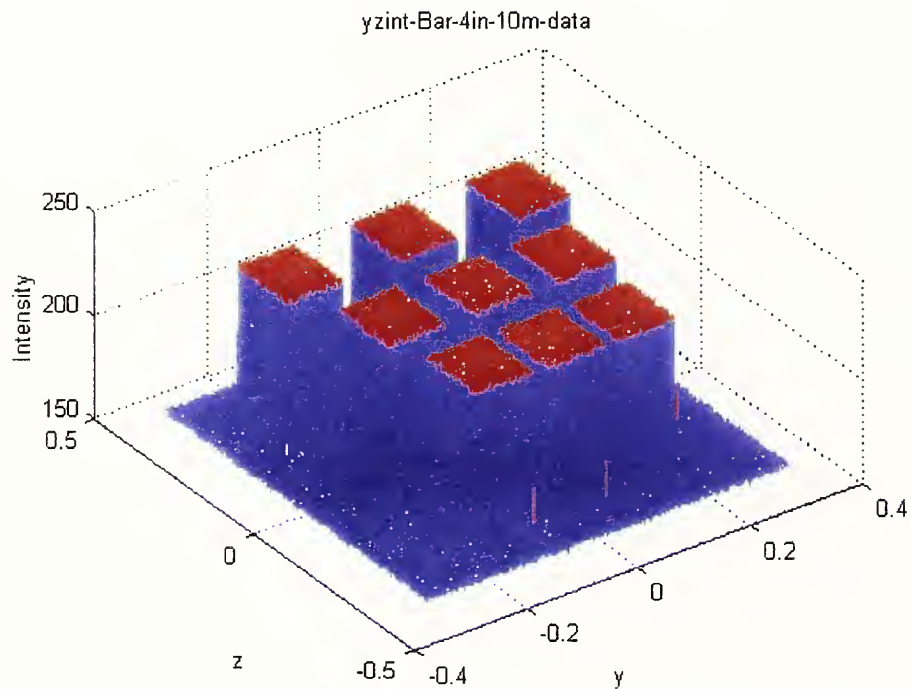
Figure 5.4. A 3D plot of the simulated 0.1016 m (4 in) bar configuration. It shows the slight variability of the data around the mean.

5.2 Creating Simulated Distorted Bar Code Images

The first computation performed was to use the simulated ground truth data sets and several models of the beam spread functions to generate blurred images. This involved a straightforward convolution calculation where the simulated bar code images played the part of the F function and the blurring models played the part of the H function.

5.2.1 Simulated Three Beam Model

Based upon the measurements of the LADAR beam described in Section 2, three models of the beam were constructed with one each for 10 m, 20 m and 40 m. For 10 m a single averaging filter was created and for 20 m and 40 m two beam models were constructed of three vertical averaging filters each. All of the beam models were defined in terms of discrete points with grid spacing the same as the grid spacing of the ground truth data sets. Each of the discretized beams was specified in terms of a square array. The 10 m discretized beam was taken as a 15 x 15 array, the 20 m discretized beam was a 17 x 17 array and the 40 m discretized beam was a 29 x 29 array. Each of the mesh spacings was taken as approximately 3 mm, which was the mesh spacing of the simulated ground truth data. For all practical purposes, it was assumed that the beam model meshes and the ground truth meshes were identical.

For the 10 m beam model, a subset of the 15 x 15 grid was selected. Nonzero values were assigned to a subgrid of 8 vertical points by 15 horizontal points that fell across the middle of the 15 x 15 grid. This subgrid simulated the approximate 27 mm x 55 mm beam at 10 m and comprised 120 points. The nonzero value assigned to each subgrid point was 1/120 so that the beam model would act like an averaging filter in the convolution process.

The 20 m and 40 m beam models were each made up of three submatrix averaging filters to simulate the split beams as described in Section 2. For the 20 m model, the grid spacing was divided so that there were three 15 x 4 nonzero subgrids, far left, middle, and far right, separated by two approximately equal zero subgids. Since there were 180 nonzero grid points, 1/180 was assigned to each point. For the 40 m model three nonzero subgrids were selected, a 29 x 5 to the far left, a 29 x 7 in the middle and a 29 x 5 to the far right with zero assigned elsewhere. Since there were 493 nonzero grid points a value of 1/493 was assigned to each of them.

5.2.2 Simulated Blurred Bar Codes

In this section we show the results of using the three simulated beam models as convolution kernels applied to the simulated ground truth bar codes. The blurred images were compared with the actual measured bar code images. This comparison is shown in Figure 5.5 for the 0.0254 m (1 in) bars, in Figure 5.6 for the 0.0508 m (2 in) bars, and in Figure 5.7 for the 0.1016 m (4 in) bars.
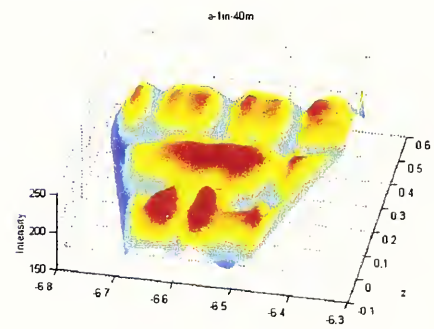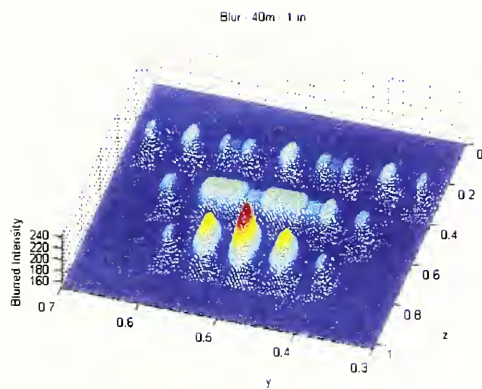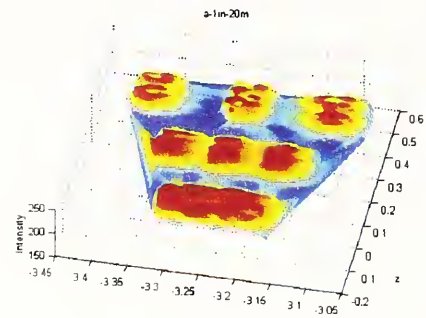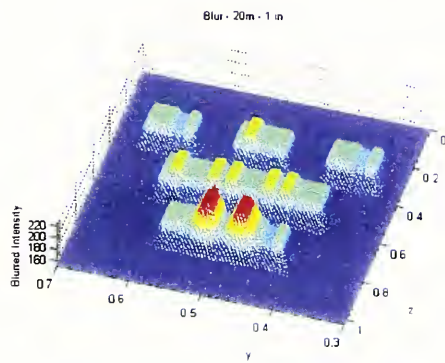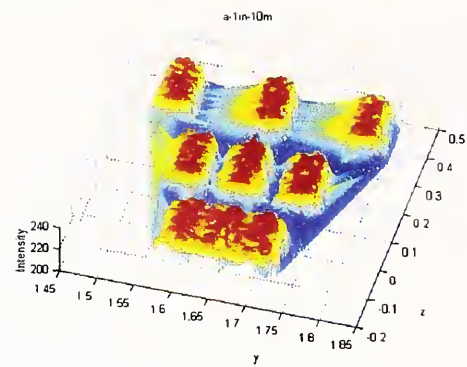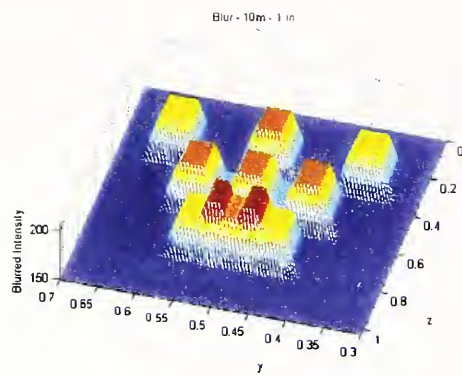
Simulated Blurring

Actual Bar Images

Figure 5.5. Comparison of Simulated Blurring of 0.0254 m (1 in) Bar Codes at 10 m, 20 m and 40 m with measured Bar Codes.
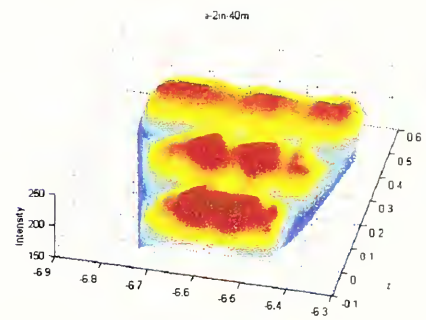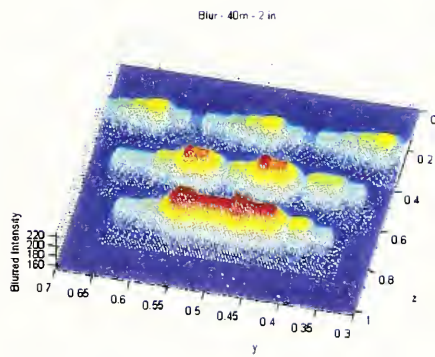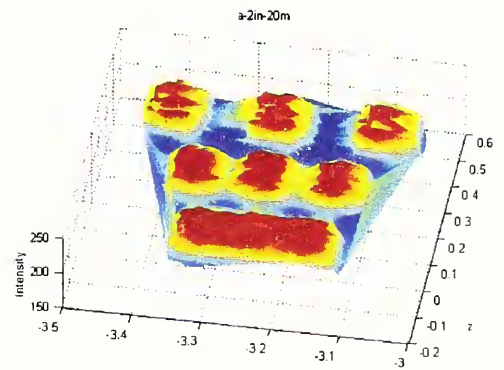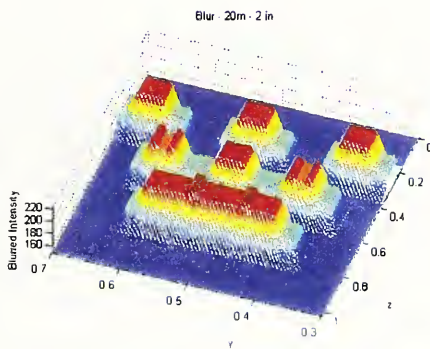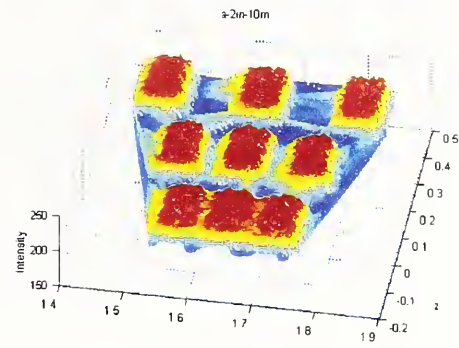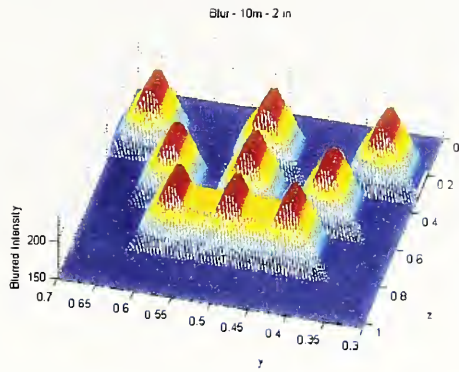
Simulated Blurring   Actual Bar Images

Figure 5.6. Comparison of Simulated Blurring of 0.0508 m (2 in) Bar Codes at 10 m, 20 m and 40 m with measured Bar Codes.
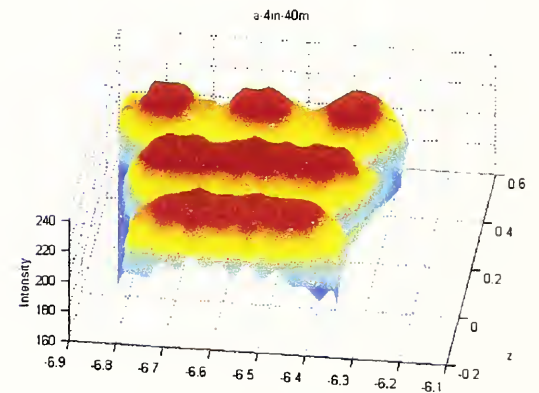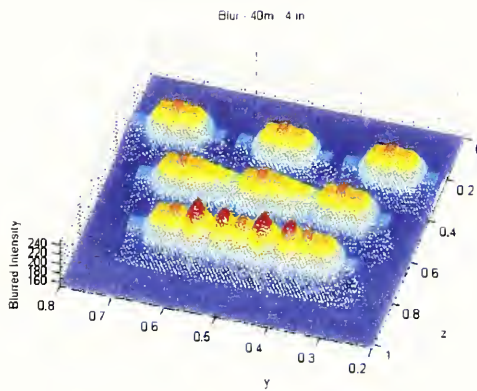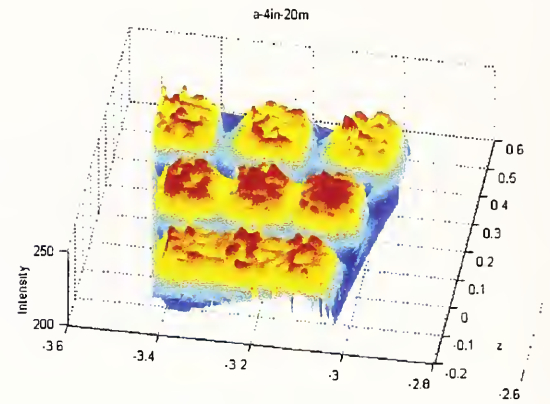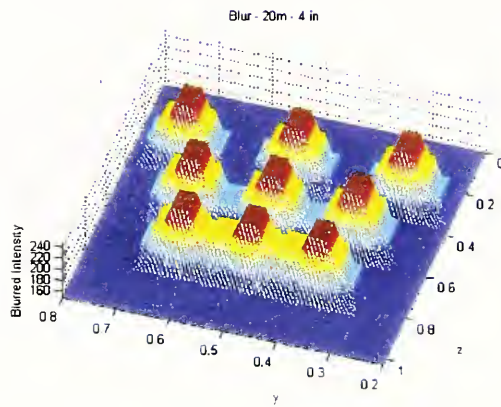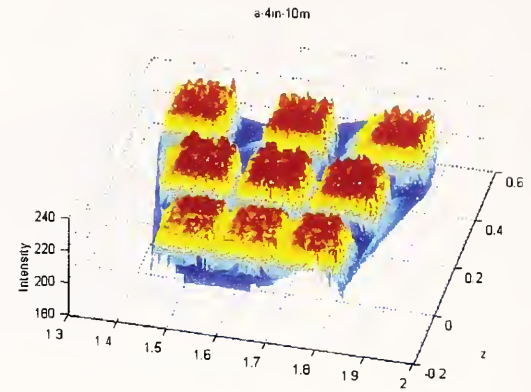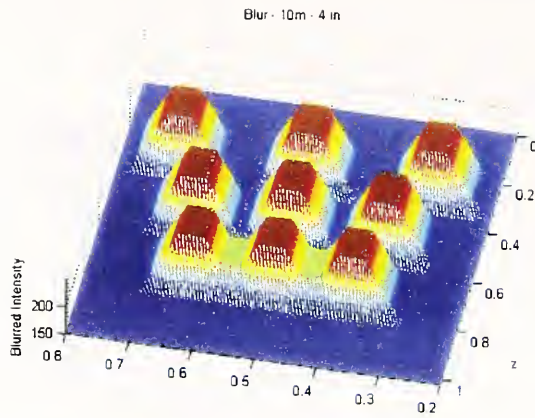
Simulated Blurring   Actual Bar Images

Figure 5.7. Comparison of Simulated Blurring of 0.1016 m (4 in) Bar Codes at 10 m, 20 m and 40 m with measured Bar Codes.

48

The first observation that can be made about the differences of the simulated blurred LADAR images and the measured LADAR images is that the measured images appear broader horizontally than the simulated images. Due to the convolution integral the simulated blurring process is only a local averaging. The figure does show that the beam model does average together portions of neighboring bars. This also appears to happen with the measured data, but the measured data appears to be broader horizontally. This suggests that there are nonlinear components to the LADAR data acquisition process in addition to local averaging. Some simple measurements of the blurred data at 10 m indicated that the 0.0254 m (1 in) bar measured data spread horizontally to approximately 0.07112 m (2.8 in) whereas vertically they did not spread beyond 0.1524 m (6 in). The 0.0508 m (2 in) bars expanded horizontally to approximately 0.1016 m (4 in) and the 0.1016 m (4 in) bars expanded to approximately 0.1524 m (6 in). Again no appreciable vertical expansion was noted. This clearly indicated that a more complex model than the linear deconvolution model would probably be required to do a complete deconvolution.

Next we can examine the groupings by bar size. For the 0.0254 m (1 in) bars the simulated averaging filter at 10 m tends to produce peaks in the lower bars in the middle between the two bars. This is a known phenomenon in imaging. An averaging filter that overlaps two objects separated by a low intensity region will produce a peak between the two objects. This phenomenon does not appear in the measured image. Thus the effect of the beam on the bar code artifacts involves more than averaging. How the photons are reflected and processed by the LADAR is an open question that requires specialized metrology devices not currently available. At 20 m and 40 m, the multiple beam models do seem to produce simulated blurred results that have some relation to the measured data. Again all of the measured data show a horizontal broadening effect.

For the 0.0508 m (2 in) simulated bars, the blurring again produces sharper edged results than the measured responses. We should note that at 40 m the simulated and measured results seem to have similar peaks suggesting again that the split beam model is likely to be a candidate portion of the true blurring process. At 20 m however the split beam model does not appear to produce the correct peak locations. This is also reflected in the 0.1015 m (4 in) bar images, whereas the 10 m and 40 m results seem to be related.

All of these results indicate that the beam models selected are weak approximations of the true imaging process. However, until both a more theoretical analysis of the LADAR imaging process and more detailed metrology devices are available for the study of LADARs, we have to depend on the approximate models. These approximate models will reveal below the extent to which ground truth can be recovered from the measured images of the bar code artifacts.

5.3 Ground Truth Reconstruction Using Various Beam Spread Functions

In this section we cover the results obtained through reconstruction or deconvolution calculations using two classes of beam spread functions. The first set is based on the

beam spread functions used in the previous section to simulate blurred bar codes. The second set is based on beam spread functions created from the measured spot data.

5.3.1 Reconstruction Using Averaging Filters

Figure 5.8 shows the possibility of deconvolving the blurred 0.0254 m (1 in) and 0.0508 m (2 in) bars at 10 m. In both cases the simple solid averaging filter was used. As can be seen, the lower group of three bars has been separated to the point that they could be identified.
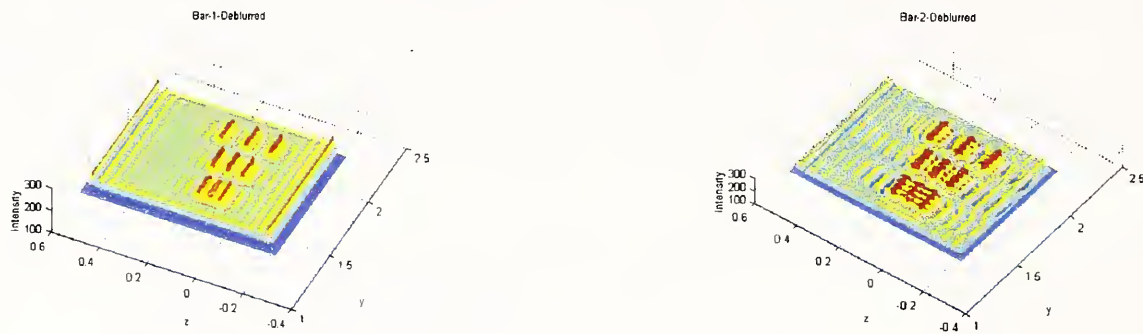
Figure 5.8. Reconstruction by a 15 x 15 Averaging Filter of the Measured Bar Codes at 10 m for the 0.0254 m (1 in) and 0.0508 (2 in) Bars.

These reconstructions were developed using the averaging filter at 10 m defined in Section 5.2.1. However, the reconstructions of different bar code sizes are very sensitive to the size of the filter. Figs. 5.9 and 5.10 show the results of reconstruction calculations for the 0.0254 m (1 in) and 0.0508 (2 in) bars at 10 m based on 11 row by 11 column averaging filters. Several filter forms were experimented with in order to determine potential characteristics of optimum deconvolution filters for these cases. Fig. 5.9 shows the results of several averaging filters applied to reconstruct 0.0254 m (1 in) bar codes. The filters were constructed from an 11 x 11 base filter in which only a few rows, centered about the middle of the 11 x 11 filter, were set to non-zero values while all of the rest of the filter was set to zero. These are called *submatrix filters* in Figs. 5.9 and 5.10. It is clear that thinner filters, in the sense of fewer non-zero rows, relative to the number of columns provide better deconvolution filters for the cases studied. This would seem appropriate since the LADAR used spread images more horizontally rather than vertically. One would expect that a filter with a height to width ratio less than one would be a reasonable selection. Figure 5.10 shows the results of applying the same sequence of filters to the 0.0508 m (2 in) measured bar data. None of the filters was successful in deconvolving these images. The conclusion that could be drawn here was that filters used to reconstruct the 0.0254 m (1 in) bar code images at 10 m could not be used to deconvolve the 0.0508 m (2 in) bars at 10 m. That is, a filter that deconvolves one image may not deconvolve another of similar form but of different sized image objects. Even though Fig. 5.10 shows the lack of reconstruction it also shows that as the height to width ratio increases the reconstruction becomes worse. This no doubt reflects the fact that the
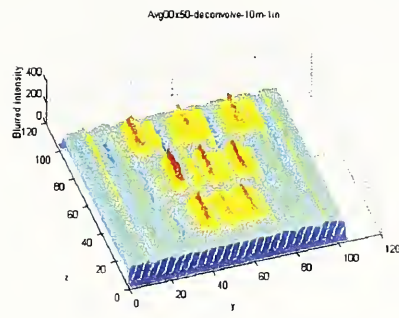
1 x 11 submatrix filter
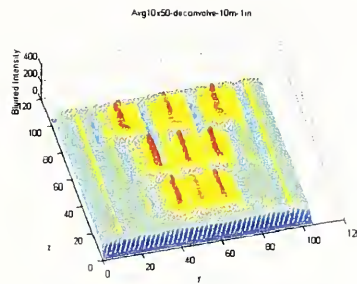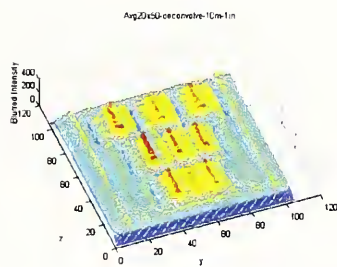
3 x 11 submatix filter

5 x 11 submatrix filter
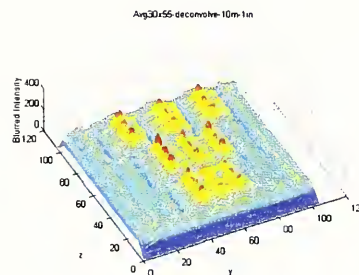
7 x 11 submatrix filter
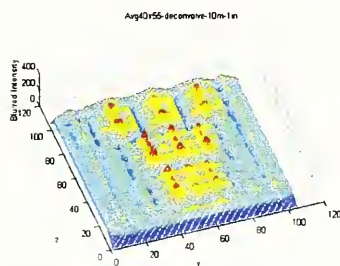
9 x 11 submatrix filter
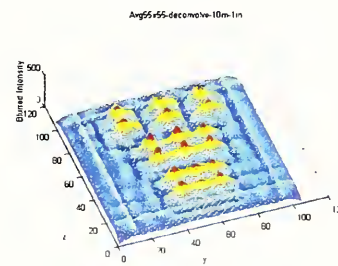
11 x 11 submatrix filter

Figure 5.9. This figure shows the effect of applying different sized filters to reconstructing 0.0254 m (1 in.) bar codes.
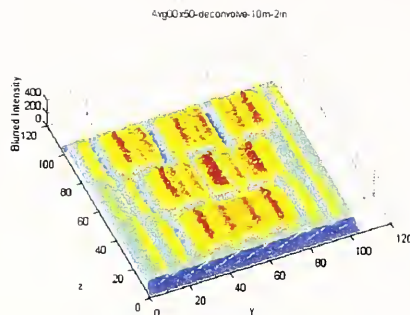
1 x 11 submatrix filter
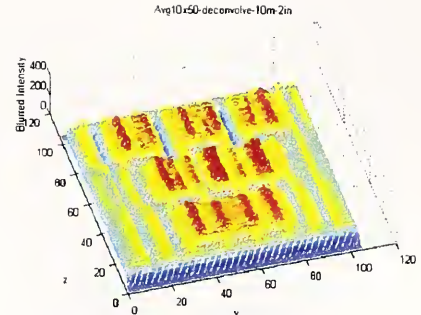
3 x 11 submatrix filter
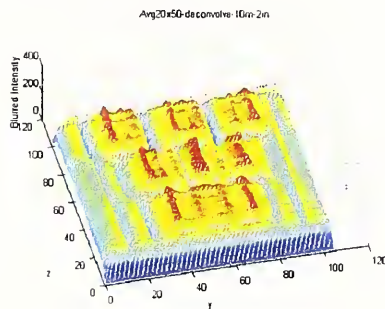
5 x 11 submatrix filter
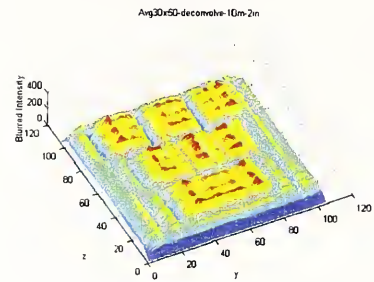
7 x 11 submatrix filter
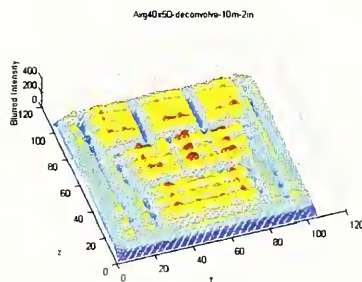
9 x 11 submatrix filter
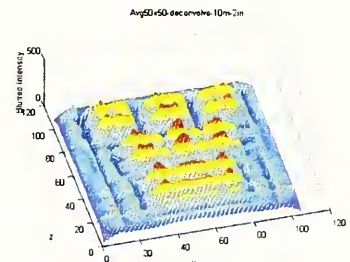
11 x 11 submatrix filter

Figure 5.10. This figure shows the effect of applying different sized filters to reconstructing 0.0508 m (2 in.) bar codes.

LADAR spreads data more horizontally than vertically. Figs. 5.9 and 5.10 were created after 10 iterations of the deconvolution algorithm. More iterations did not necessarily create a better reconstructed image. This will be shown in a figure below.

5.3.2 Reconstruction Using Filters Based on Spot Reflections

The data used to develop the beam spread functions used for these calculations was described in Section 2.5.2. Fig. 5.11 shows a sample distribution of the spread function model for the 0.00635 m (1/4 in) spot data. A similar distribution was developed for the 0.0127 m (1/2 in) spot data. Both were developed from data acquired at 10 m. As Figs. 5.12 and 5.13 show, the filters constructed based on the spot data did not produce successful bar code reconstructions at 10 m. The reconstruction using the smaller reflection point data appears more successful than that for the larger spot data. Furthermore, as shown in Fig. 5.12, a 3 x 11 element matrix filter produced the best results. Thus the 11 x 11 matrix with a submatrix of 3 non-zero rows from the 0.00635 m (1/4 in) spot data produced somewhat better results than did a submatrix with the full 11 non-zero rows.
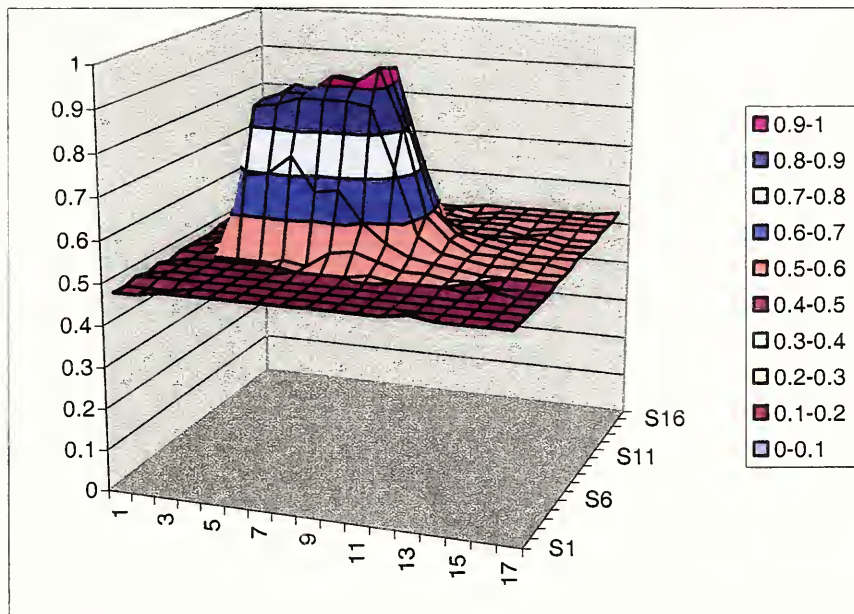
Figure 5.11. Sample plot of the data distribution from a 0.00635 m (1/4 in.) spot at 10 m.
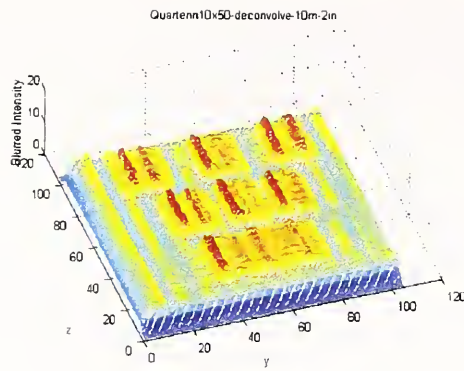
3 x 11 submatrix filter
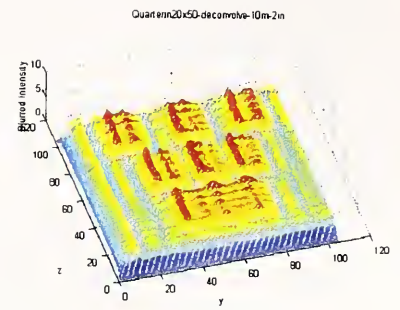
5 x 11 submatrix filter
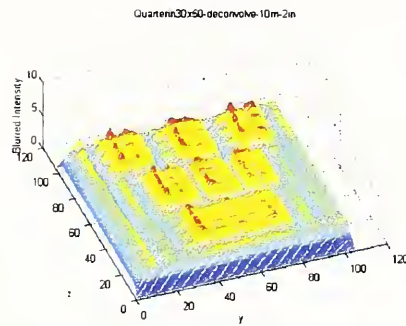
7 x 11 submatrix filter
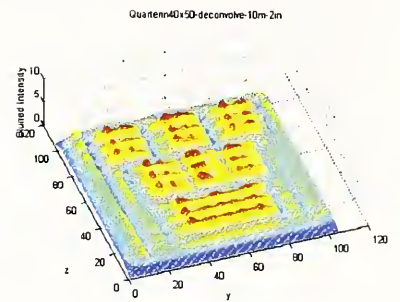
9 x 11 submatrix filter
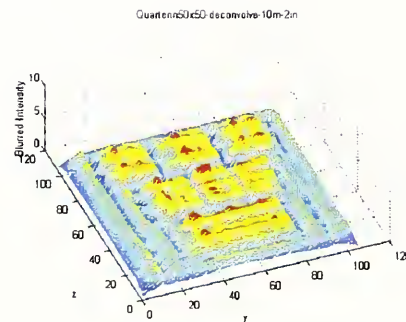
11 x 11 submatrix filter

Figure 5.12. Reconstruction results of 0.0508 m (2 in) bar codes at 10 m using filters based on data from 0.00635 m (1/4 in.) reflected spot data.

Halfin-deconvolve-10m-1in
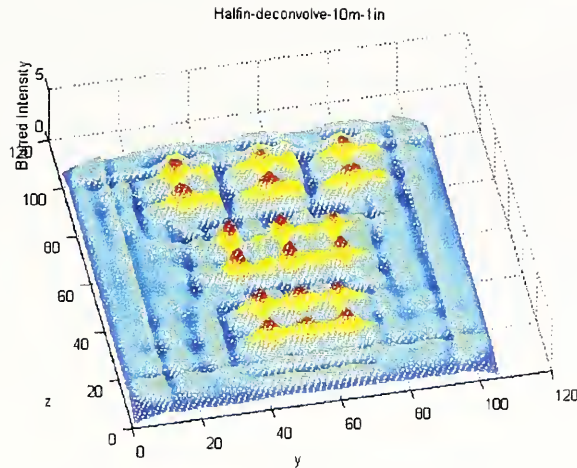
Figure 5.13. Reconstruction of 0.0508 m (2 in) bars with data from 0.0127 m (1/2 in) spot data using an 11 x 11 submatrix filter.

5.3.3 Convergence Aspects of the Reconstruction Process

When the deconvolution algorithm converged to a solution, it did so in only a few iterations. Allowing the algorithm to proceed for more iterations tended to degrade the solution image. This is demonstrated in Figs. 5.14 and 5.15.

Figure 5.14 shows the progress of the deconvolution process on the measured 0.0254 m (1 in) bars at 10 m. The figure shows iterations 1, 2, 5, 6, 9 and 10. The lower three bars are beginning to be reconstructed by iterations 5 and 6 and are essentially recovered by iteration 10. This was the general experience of applying the Paige and Saunders algorithm. That is, when there is convergence, it occurs rapidly. These results were achieved using the flat filter.

Figure 5.15 shows the results of allowing the algorithm to proceed to 100 iterations using the 0.00635 m (1/4 in) spot data. The result can be compared with Fig. 5.12 to show that adding more iterations tends to degrade results. Possible reasons for this degradation the accumulation of round off and increased oscillation in the solution image. This can be observed in Fig. 5.14. Note that as the iteration progresses more surface oscillations become visible.
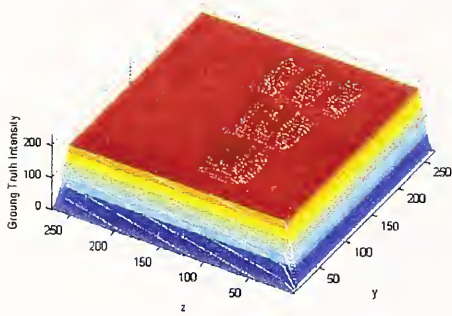
Iteration 1 for 0.0254 m (1 in) bars
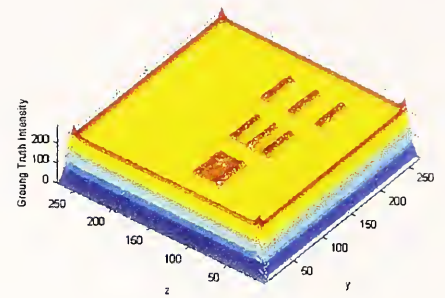
Iteration 2 for 0.0254 m (1 in) bars
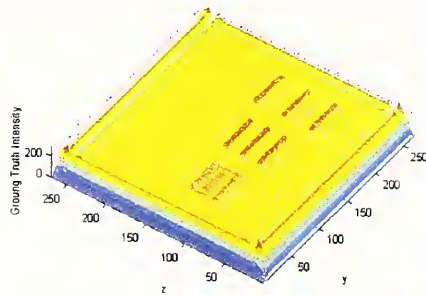
Iteration 5 for 0.0254 m (1 in) bars
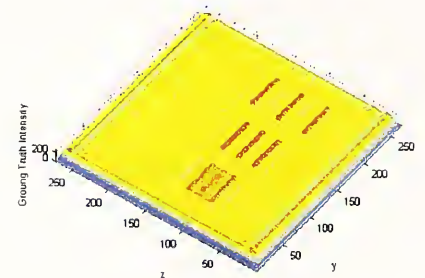
Iteration 6 for 0.0254 m (1 in) bars
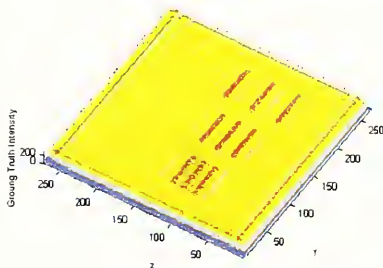
Iteration 9 for 0.0254 m (1 in) bars
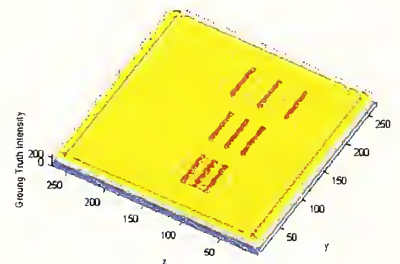
Iteration 10 for 0.0254 m (1 in) bars

Figure 5.14. This figure shows sample iteration results of the Paige and Saunders algorithm used to deconvolve the measured 0.0254 m (1 in) bars at 10 m.
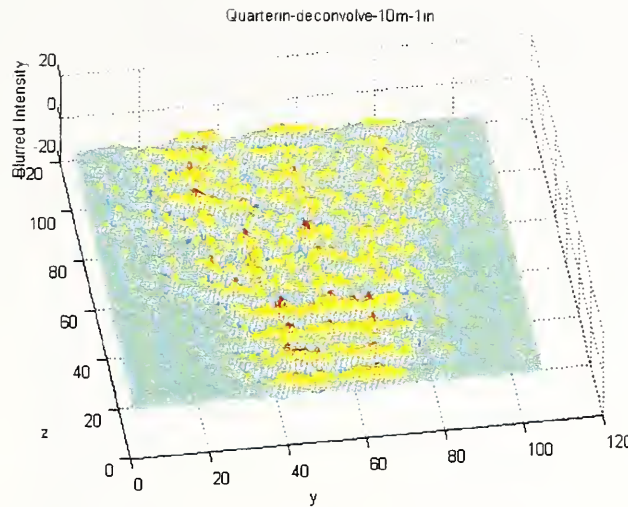
Figure 5.15. 100 iterations of the deconvolution algorithm.

5.3.4 Reverse Engineering Filter Construction

One approach to constructing a deconvolving filter is to reverse the position of the ground truth vector and the filter matrix. That is, one assumes that the ground truth is known and that the filter is unknown. Thus the general problem of solving for the filter becomes one of solving $G = FH$ where G is the blurred image and F is the assumed ground truth put into a matrix form while the unknown filter is treated as an unknown vector H. A sample program for performing the reverse engineering filter identification is given in Appendix C5. The essential idea in the subroutine Aprod was to set the indices up in such a manner that the ground truth matrix aligned with the filter matrix, treated as a vector. A discussion of how these matrices were aligned is also given in Appendix C5.1.

When the program was implemented on the simulated 0.0254 m (1 in) bar data, it produced a filter essentially the same size as the blurred image but the product of the ground truth matrix and the reverse engineered filter produced a blurred image nearly indistinguishable from the blurred image. This is shown in Fig. 5.16.
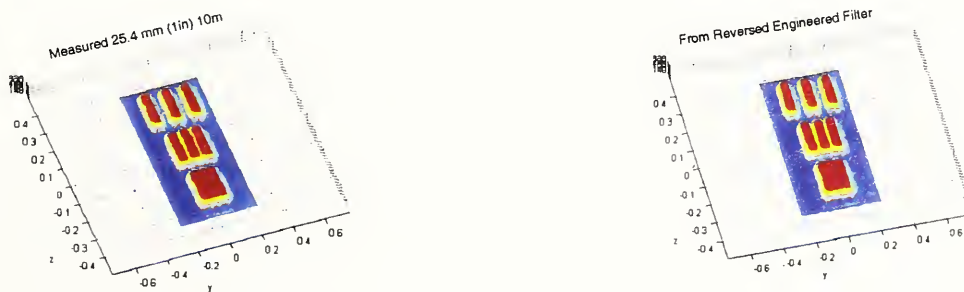
Figure 5.16. Results of applying the reverse engineered filter to ground truth data for 0.0254 m (1 in) bars.

Even though the reverse engineering algorithm appeared to produce a filter that could reproduce the original blurred data when applied to the simulated ground truth, when the filter was used in the deconvolution progam it was unsuccessful in reproducing the ground truth image. This is shown in Fig. 5.17.
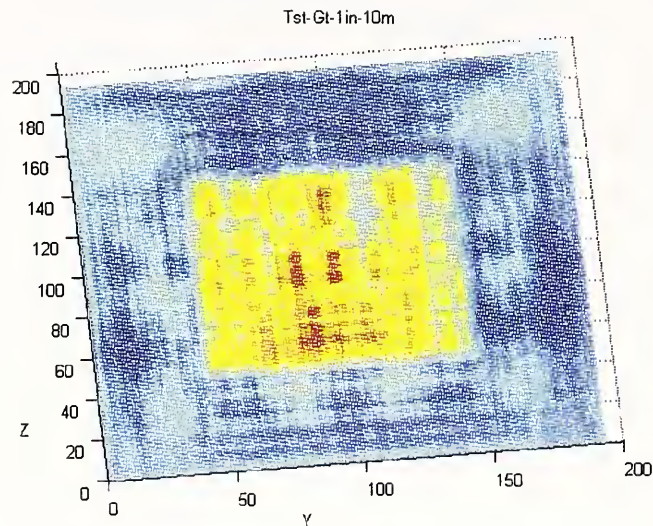
Figure 5.17. Result of applying the reverse engineered filter in the deconvolution program using the 0.0254 m (1 in) bar measured data.

6.0 Summary and Future Directions

6.1 Summary

The applications for LADARs, ranging from terrain representation to object recognition, have been increasing steadily over the past two decades and continue to grow. As compared to traditional methods, LADARs enable the rapid capture of large amounts of 3D information- several million points per scan. A typical LADAR locates a point via range and angular information, providing coordinates centered at the instrument. In addition to range and angular information, most LADARs return an intensity level that can be used for imaging. Some LADARs also return other spectral data such a red, green, and blue color levels.

LADARs have been used at NIST for several projects. One of these projects, involved the use of LADARs for object recognition at construction sites. Object recognition is crucial from several standpoints: automating processes such as "pick and place", ground truth determination, inventory, and obstacle avoidance. The wide and ubiquitous use of bar codes led to the possibility of a similar implementation for a construction site. The idea was to use the intensity information from a LADAR to "read" a bar code. Experiments were conducted to determine the feasibility of this idea.

These experiments involved scanning bar codes made of highly reflective material in various configurations and at various distances. The data obtained by the LADAR indicated that beyond 10 m the images of the bar become so distorted that image processing techniques would have to be applied to even attempt to recover the bar configurations. However, the image processing techniques require an understanding of how the LADAR beam dispersed and was reflected, and how the return signal is processed by the LADAR optics. Much of the information about data processing by the LADAR is usually proprietary to the LADAR manufacturer.

An attempt was made to create an approximate model of these processes, called a Beam Spread Function. This beam spread function was needed to attempt to reconstruct or deconvolve the bar configurations from the distorted images. The deconvolution algorithm used depended on an iterative solution method for a large least squares minimization problem in which the distorted LADAR image was approximated by the product of a large matrix representing the Beam Spread Function and the unknown ground truth image. The Beam Spread Function model was represented by a large sparse matrix. In order to minimize computation time in the iterative least squares algorithm a fast sparse matrix – vector multiply algorithm was developed that also minimized storage requirements for the matices and vectors involved.

Although deconvolution is often highly successful in reconstructing photographic images, the results of the current study indicate that until finer resolution LADARs are made available, deconvolution techniques have limited capability of recovering LADAR images.

6.2. Future Work

The methods used to approximate the beam spread function were very crude as there were no procedures, standard or otherwise, and facilities available for this purpose. These experiments and others to characterize a LADAR underscore the necessity of an intramural test facility. In keeping with its mission as the Nation's metrology laboratory, NIST is in a position to provide such metrology support to both users and manufacturers of LADARs in addition to meeting its own substantial internal calibration needs. As a first step towards establishing this test facility, a LADAR calibration facility workshop was convened in June, 2003. It is envisioned that such a facility would enable characterization of LADARs in terms of range and pointing angle uncertainties, beam spread, and resolution.

7.0 References

[1] Cheok, G. S., Leigh, S., and Rukhin, A., "Calibration Experiments of a Laser Scanner", NISTIR 6922, National Institute of Standards and Technology, Gaithersburg, MD, September, 2002.

[2] Elden, L. "Algorithms for the regularization of ill-conditioned least squares problems", BIT, Vol. 17, 1977, 134-145.

[3] Gilsinn, D. E., Cheok, G. S., O'Leary, D. P., "Reconstructing images of bar codes for construction site recognition", NIST SP 989, 19th International Symposium on Automation and Robotics in Construction, NIST, Gaithersburg, MD, September, 23-25, 2002, 363-368.

[4] Golub, G., "Numerical methods for solving linear least squares problems", Numerische Mathematik, Vol. 7, 1965, 206-216.

[5] Golub, G., Kahan, W., "Calculating the singular values and pseudo-inverse of a matrix", J. SIAM Numer. Anal., Ser. B, Vol. 2, No. 2, 1965, 205-224.

[6] Paige, C. "Bidiagonalization of matrices and solution of linear equations", SIAM J. Numer. Anal., Vol. 11, No. 1, 1974, 197-209

[7] Paige, C., Saunders, M., "LSQR: An algorithm for sparse linear equations and sparse least squares", ACM Transactions on Mathematical Software, Vol. 8, No. 1, 1982, 43-71

[8] Paige, C., Saunders, M., "Algorithm 583, LSQR: Sparse linear equations and least squares problems", ACM Transactions on Mathematical Software, Vol. 8, No. 2, 1982, 195-209.

[9] Saunders, M., "Solution of sparse rectangular systems using LSQR and CRAIG", BIT, Vol. 35, 1995, 588-604.

APPENDIX A: Fast Matrix Vector Products

The least squares algorithm outlined in Section 6.2 and detailed below in Appendix B required the calculation of matrix products of the form Hf and $H^T g$. The algorithms for both of these products are intimately related. They assume that a ground truth image, the distorted image and beam matrix are specified as in Section 5.

A.1 Calculating Hf

The object of the algorithm to compute the distorted image $g = Hf$ is to perform the operation by only storing the ground truth image, f, and beam spread matrix, which is only of size ma x ma. That is, compute the ng x ng distorted image by only storing the ma x ma beam spread matrix and the nf x nf ground truth image.

One begins by storing the distorted image g and the ground truth image f as vectors by columns

$$g = \begin{pmatrix} g(1,1) \\ g(2,1) \\ \vdots \\ g(ng,1) \\ g(1,2) \\ \vdots \\ g(ng,2) \\ \vdots \\ g(1,ng) \\ \vdots \\ g(ng,ng) \end{pmatrix} \qquad f = \begin{pmatrix} f(1,1) \\ f(2,1) \\ \vdots \\ f(nf,1) \\ f(1,2) \\ \vdots \\ f(nf,2) \\ \vdots \\ f(1,nf) \\ \vdots \\ f(nf,nf) \end{pmatrix} \qquad (A.1)$$

With f and g stored as vectors an examination of the discrete convolution double summation shows that the calculation would require the beam spread function be stored as a large sparse matrix of the form

$$H = \begin{bmatrix} H_1 & H_2 & \cdots & H_{ma} & 0 & \cdots & 0 \\ 0 & H_1 & H_2 & \cdots & H_{ma} & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & H_1 & H_2 & \cdots & H_{ma} \end{bmatrix} \qquad (A.2)$$

where, for $q = 1, \ldots, ma$

$$H_q = \begin{pmatrix} H(ma,ma-q+1) & H(ma-1,ma-q+1) & \cdots & H(1,ma-q+1) & 0 & \cdots & 0 \\ 0 & H(ma,ma-q+1) & H(ma-1,ma-q+1) & \cdots & H(1,ma-q+1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & H(ma,ma-q+1) & H(ma-1,ma-q+1) & \cdots & H(1,ma-q+1) \end{pmatrix} \quad (A.3)$$

Each block H_q is made up of ng rows by nf columns. The entire H matrix is made up of ng^2 rows by nf^2 columns. Each line of block H_q is made up of the same values but shifted by one for each row. The entire H matrix is made up of the same block rows but shifted by one block at the next block row. The significant thing about this structure is that one does not need to store the entire matrix. In fact one only needs to store the $ma \times ma$ beam spread matrix.

Since f and g are stored by column one can re-index them as a single dimension array $f(k) = f(i, j)$ where $k = (j-1)*nf + i$ and $g(l) = g(r,s)$ where $l = (s-1)*ng + r$. Next write f and g as blocks of vectors

$$f = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{nf} \end{pmatrix} \qquad g = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{ng} \end{pmatrix} \qquad (A.4)$$

where

$$f_i = \begin{pmatrix} f((i-1)*nf+1) \\ f((i-1)*nf+2) \\ \vdots \\ f(i*nf) \end{pmatrix} \qquad g_p = \begin{pmatrix} g((s-1)*ng+1) \\ g((s-1)*ng+2) \\ \vdots \\ g(s*ng) \end{pmatrix} \qquad (A.5)$$

The algorithm generates the g vector a block vector g_p at a time where $p = 1, 2, ..., ng$ and

$$g_p = H_1 f_p + H_2 f_{p+1} + \cdots + H_{ma} f_{p+(ma-1)} \qquad (A.6)$$

Given a p one now steps across the f_i blocks by letting $i = p, ..., p+(ma-1)$. To do this introduce an index $k = 0, 1, ..., ma-1$ and set $i = p+k$ and observe that the product $H_{k+1} f_{p+k}$ looks like

$$H_{k+1} f_{p+k} = \begin{pmatrix} H(ma, ma-k) & \cdots & H(1, ma-k) & 0 & \cdots & 0 \\ 0 & H(ma, ma-k) & \cdots & H(1, ma-k) & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & H(ma, ma-k) & \cdots & H(1, ma-k) \end{pmatrix} \begin{pmatrix} f((p+k-1)*nf+1) \\ f((p+k-1)*nf+2) \\ \vdots \\ f((p+k)*nf) \end{pmatrix} \qquad (A.7)$$

where one uses the fact that $ma-k = ma-(k+1)+1$.

The algorithm proceeds as follows: Step through $p = 1, 2, ..., ng$, i.e. loop on each g vector block. For each p, step through $l = 1, 2, ..., ng$, i.e. loop on each row of the pth vector block. Compute the current global element of the g vector that is being computed as $j = (p-1)*ng + l$. Form the double sum below.

$$g(j) = \sum_{k=0}^{ma-1} \sum_{q=0}^{ma-1} H(ma-q, ma-k) * f((p+k-1)*nf+l+q) \qquad (A.8)$$

This algorithm can be implemented with 4 loops and 13 lines of code as shown in the sample FORTRAN 90 code below.

```
DO p = 1,ng                  ! Loop over each y block of ng elements
      DO l = 1,ng            ! Loop over local row number within y block
            j = (p-1)*ng + l
            sum = 0.0
            DO k = 0,ma-1              ! Loop over f blocks of nf elements
                  DO q = 0,ma-1          ! Form the inner product for row j
                        e = (p + k - 1)*nf +l + q
                        sum = sum + h(ma-q,ma-k)*f(e)
                  END DO
            END DO
            g(j) = g(j) + sum        ! Add A*f to g for row j
      END DO
END DO
```

A.2 Calculating $H^T g$

The algorithm to compute $H^T g$ begins by setting in vector block form

$$
f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{nf} \end{bmatrix} \qquad g = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{ng} \end{bmatrix} \qquad (A.9)
$$

as in Equation (A.4). The transpose of H looks like

$$
H^T = \begin{bmatrix}
H_1^T & 0 & \cdots & 0 \\
H_2^T & H_1^T & \ddots & 0 \\
\vdots & H_2^T & \ddots & 0 \\
H_{ma}^T & \vdots & \ddots & H_1^T \\
0 & H_{ma}^T & \ddots & H_2^T \\
\vdots & \ddots & \ddots & \vdots \\
0 & 0 & 0 & H_{ma}^T
\end{bmatrix} \qquad (A.10)
$$

The computation $f = H^T g$ can be written in columnd form as

$$
f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{nf} \end{bmatrix} = H^T g = \begin{bmatrix} H_1^T \\ H_2^T \\ \vdots \\ H_{ma}^T \\ 0 \\ \vdots \\ 0 \end{bmatrix} g_1 + \begin{bmatrix} 0 \\ H_1^T \\ H_2^T \\ \vdots \\ H_{ma}^T \\ \vdots \\ 0 \end{bmatrix} g_2 + \cdots + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ H_1^T \\ H_2^T \\ \vdots \\ H_{ma}^T \end{bmatrix} g_{ng}
$$

For the sake of the current discussion initialize $f = 0$. Then the left hand side can be computed by iterative block summations. We observe that $nf = ng + ma - 1$. Thus for each g block $p = 1, 2, \ldots, ng$

Block 1

$$f_1 = f_1 + H_1^T g_1$$
$$f_2 = f_2 + H_2^T g_1$$
$$\vdots$$
$$f_{ma} = f_{ma} + H_{ma}^T g_1$$

Block 2

$$f_2 = f_2 + H_1^T g_2$$
$$f_3 = f_3 + H_2^T g_2$$
$$\vdots$$
$$f_{ma+1} = f_{ma+1} + H_{ma}^T g_2$$

Block p

$$f_p = f_p + H_1^T g_p$$
$$f_{p+1} = f_{p+1} + H_2^T g_p$$
$$\vdots$$
$$f_{p+(ma-1)} = f_{p+(ma-1)} + H_{ma}^T g_p$$

Block ng

$$f_{ng} = f_{ng} + H_1^T g_{ng}$$
$$f_{ng+1} = f_{ng+1} + H_2^T g_{ng}$$
$$\vdots$$
$$f_{nf} = f_{ng+(ma-1)} = f_{ng+(ma-1)} + H_{ma}^T g_{ng}$$

This sequence of block operations constitutes the outer loop of the algorithm. For the pth block in this loop one must compute individual elements on the left using elements on the right. This constitutes the first inner loop where each local index of g_p is associated with the global index used to define the entire vector g. Thus, for each $l = 1, 2, ..., ng$ associate the global index $j = (p-1)ng + l$. Therefore, the pth block of g can be written

$$g_p = \begin{bmatrix} g((p-1)ng+1) \\ \vdots \\ g((p-1)ng+l) \\ \vdots \\ g((p-1)ng+ng) \end{bmatrix}$$

Note that in each p block there are ma separate summations given by

$$f_{p+k} = f_{p+k} + H_{k+1}^T g_p$$

for $k = 0, 1, \ldots, ma\text{-}1$ which constitutes the second inner loop. The matrix-vector product $H_{k+1}^T g_p$ is evaluated as an accumulated summation

$$H_{k+1}^T g_p = \begin{bmatrix} h(ma, ma-k) \\ \vdots \\ h(1, ma-k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} g((p-1)ng+1) + \cdots + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ h(ma, ma-k) \\ \vdots \\ h(1, ma-k) \end{bmatrix} g((p-1)ng+ng)$$

$$= \begin{bmatrix} h(ma-0, ma-k) \\ \vdots \\ h(ma-q, ma-k) \\ \vdots \\ h(ma-(ma-1), ma-k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} g((p-1)ng+1) + \cdots$$

$$+ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ h(ma-0, ma-k) \\ \vdots \\ h(ma-q, ma-k) \\ \vdots \\ h(ma-(ma-1), ma-k) \end{bmatrix} g((p-1)ng+ng)$$

where the index $q = 0, 1, \ldots, ma\text{-}1$ represents the index of the final inner loop. The left hand vector f_{p+k} is given by

$$f_{p+k} = \begin{bmatrix} f((p+k-1)nf + 1) \\ \vdots \\ f((p+k-1)nf + nf) \end{bmatrix}$$

Note that the entire summation above is identified by two indices. The index $l = 1, 2, \ldots, ng$ identifies the term in the summation. The index $q = 0, 1, \ldots, ma\text{-}1$ identifies the row for the given term l. If we assume that the vector f_{p+k} has been initialized before the summation begins then the sum can be accumulated as

$$f((p+k-1)nf + l + q) = f((p+k-1)nf + l + q) + h(ma - q, ma - k)g((p-1)ng + l)$$

This summation proceeds as follows. Beginning with the $l = 1$ term the first ma rows (i.e. $q = 0, 1, \ldots, ma\text{-}1$) are added to the initialized first ma elements of f_{p+k} (i.e. $l + q = 1, 2, \ldots, ma$). Then the elements of rows 2, 3, $\ldots$, $ma + 1$ of the $l = 2$ term are added to the 2, 3, $\ldots$, $ma + 1$ elements of f_{p+k} (i.e. $l + q = 2, 3, \ldots, 2 + (ma - 1)$). Next rows 3, 4, $\ldots$, $ma + 2$ of the $l = 3$ term are added to rows 3, 4, $\ldots$, $ma + 2$ of f_{p+k} (i. e. $l + q = 3, 4, \ldots, 3 + (ma - 1)$). This process continues until $l = ng$. Note that the indexing accounts for the shifting down by one element of the columns of H_{k+1}^T for $l = 1, 2, \ldots, ng$.

In terms of the actual implementation of the code k is incremented after l is set. That is, all first terms of the sum are accumulated for $k = 0, 1, \ldots, ma - 1$, then the second terms are accumulated for each k and so forth.

```
DO p = 1,ng
      DO l = 1,ng
            j = (p-1)*ng + l
            gj = g(j)
            DO k = 0,ma-1
                  DO q = 0,ma-1
                        e = (p+k-1)*nf + l + q
                        f(e) = f(e) + h(ma-q,ma-k)*gj
                  END DO
            END DO
      END DO
END DO
```

68

APPENDIX B: LSQR Algorithm

In this appendix we present an expanded discussion of the least squares algorithm presented in Section 4. The algorithm is that of Paige and Saunders [6, 7]. The first part of the appendix describes the iterative development of the U and V matrices such that

$$H = UBV^T$$

where B is bidiagonal. The second part of the appendix describes the algorithm for the sequential minimization of a reduced least squares problem.

B.1 Computing U and V

To compute U and V, given columnwise with

$$V = [v_1, \quad v_2, \quad \cdots, \quad v_n], \quad \text{(B.1)}$$

note that

$$H[v_1, \ \cdots, \ v_i, \ \cdots, \ v_n] = [u_1, \ \cdots, \ u_i, \ u_{i+1}, \ \cdots, \ u_n] \begin{bmatrix} \alpha_1 & & & & & \\ \beta_2 & \alpha_2 & & & 0 & \\ & \ddots & \ddots & & & \\ & & \beta_i & \alpha_i & & \\ & & & \beta_{i+1} & \ddots & \\ & 0 & & & \ddots & \ddots \\ & & & & & \beta_n & \alpha_n \end{bmatrix} \quad \text{(B.2)}$$

implies that

$$Hv_i = \alpha_i u_i + \beta_{i+1} u_{i+1} \quad \text{(B.3)}$$

or to compute the $(i+1)$-st term of U let

$$\beta_{i+1} u_{i+1} = Hv_i - \alpha_i u_i \quad \text{(B.4)}$$

where β_{i+1} is selected so that $\|u_{i+1}\| = 1$. Furthermore

$$H^T [u_1, \cdots, u_{i+1}, \cdots, u_n] = [v_1, \cdots, v_i, v_{i+1}, \cdots, v_n] \begin{bmatrix} \alpha_1 & \beta_2 & & & & & \\ & \alpha_2 & \beta_3 & & & & 0 \\ & & \ddots & \ddots & & & \\ & & & & \alpha_i & \beta_{i+1} & \\ & & & & & \alpha_{i+1} & \ddots \\ & 0 & & & & \ddots & \beta_n \\ & & & & & & \alpha_n \end{bmatrix}. \quad (B.5)$$

Then

$$H^T u_{i+1} = \beta_{i+1} v_i + \alpha_{i+1} v_{i+1} \quad (B.6)$$

or to compute the $(i+1)$-st term of V let

$$\alpha_{i+1} v_{i+1} = H^T u_{i+1} - \beta_{i+1} v_i. \quad (B.7)$$

Thus one can sequentially generate the columns of U, V and the bidiagonal elements of B as follows. Initialize $u_1 = g / \beta_1$ where $\beta_1 = \|g\|$. Next compute v_1 so that $\alpha_1 v_1 = H^T u_1$ and $\|v_1\| = 1$. Then for $i = 1, 2, \ldots, n\text{-}1$ compute u_{i+1}, v_{i+1} so that $\|u_{i+1}\| = \|v_{i+1}\| = 1$ by

$$\begin{aligned} \beta_{i+1} u_{i+1} &= H v_i - \alpha_i u_i \\ \alpha_{i+1} v_{i+1} &= H^T u_{i+1} - \beta_{i+1} v_i \end{aligned} \quad . \quad (B.8)$$

At the kth stage one would have computed

$$\begin{aligned} U_k &= [u_1, \cdots, u_k] \\ V_k &= [v_1, \cdots, v_k] \\ B_k &= \begin{bmatrix} \alpha_1 & 0 \\ \beta_2 & \alpha_2 \\ & \ddots & \ddots \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix} \end{aligned} \quad (B.9)$$

where B_k is a $(k+1) \times k$ matrix. One further has

$$U_{k+1} (\beta_1 e_1) = b \quad (B.10)$$

since

$$U_{k+1}(\beta_1 e_1) = [u_1, \quad \cdots, \quad u_{k+1}] \begin{bmatrix} \beta_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \beta_1 u_1 = g. \quad (B.11)$$

It is clear that

$$HV_k = U_{k+1}B_k. \quad (B.12)$$

One can also show that

$$H^T U_{k+1} = V_k B_k^T + \alpha_{k+1} u_{k+1} e_{k+1}^T. \quad (B.13)$$

This follows from

$$\begin{aligned} H^T U_{k+1} = A^T [u_1, \quad \cdots, \quad u_{k+1}] &= [\alpha_1 v_1, \quad \beta_2 v_1 + \alpha_2 v_2, \quad \cdots, \quad \beta_{k+1} v_k + \alpha_{k+1} v_{k+1}] \\ &= [\alpha_1 v_1, \quad \beta_2 v_1 + \alpha_2 v_2, \quad \cdots, \quad \beta_{k+1} v_k] + \alpha_{k+1} v_{k+1} [0, \quad 0, \quad \cdots, \quad 1] \quad . \quad (B.14) \\ &= V_k B_k^T + \alpha_{k+1} v_{k+1} e_{k+1}^T \end{aligned}$$

B.2 The sequential minimization algorithm

Now we will explain the algorithm used to solve a sequence of minimization problems written as

$$\min_{y_k} \| B_k y_k - \beta_1 e_1 \|. \quad (B.15)$$

For notation, set

$$\begin{aligned} f_k &= V_k y_k \\ r_k &= g - A f_k \quad \quad (B.16) \\ t_{k+1} &= \beta_1 e_1 - B_k y_k \end{aligned}$$

r_k and t_{k+1} are related by

$$r_k = U_{k+1} t_{k+1} \quad (B.17)$$

since

$$r_k = g - Hx_k = U_{k+1}(\beta_1 e_1) - HV_k y_k$$
$$= U_{k+1}(\beta_1 e_1) - U_{k+1} B_k y_k$$
$$= U_{k+1}(\beta_1 e_1 - B_k y_k) \qquad \text{(B.18)}$$
$$= U_{k+1} t_{k+1}$$

Introduce the augmented matrix of columns $[B_k \quad \beta_1 e_1]$. It will be shown below that B_k can be transformed by an orthogonal matrix Q_k as follows

$$Q_k B_k = \begin{bmatrix} R_k \\ 0 \end{bmatrix} \qquad \text{(B.18)}$$

where Q_k is $(k+1) \times (k+1)$, B_k is $(k+1) \times k$. The right hand side is $(k+1) \times k$ and

$$R_k = \begin{bmatrix} \rho_1 & \theta_2 & 0 & \cdots & 0 \\ 0 & \rho_2 & \theta_3 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \theta_k \\ 0 & \cdots & \cdots & 0 & \rho_k \end{bmatrix}. \qquad \text{(B.19)}$$

To show how the orthogonal matrix Q_k is developed let

$$B = \begin{bmatrix} \alpha_1 & 0 & \cdots & \cdots & 0 \\ \beta_2 & \alpha_2 & 0 & \cdots & 0 \\ 0 & \beta_3 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \beta_n & \alpha_n \end{bmatrix}. \qquad \text{(B.20)}$$

Suppose that B has been reduced at the $(k-1)^{st}$ step to

$$\bar{B} = \begin{bmatrix} \rho_1 & \theta_2 & & & & & \\ & \rho_2 & \theta_3 & & & & \\ & & \ddots & \ddots & & & \\ & & & \rho_{k-1} & \theta_k & & \\ & & & & \alpha_k & & \\ & & & & \beta_{k+1} & \alpha_{k+1} & \\ & & & & & \ddots & \alpha_n \\ & & & & & & \beta_n \end{bmatrix}. \qquad \text{(B.21)}$$

Construct the kth Givens rotation matrix as follows.

$$\overline{Q} = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & \ddots & & & & & \\ & & & c_k & s_k & & & \\ & & & s_k & -c_k & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix} \qquad \text{(B.22)}$$

where c_k, s_k are on the k-th row and s_k, $-c_k$ are on the $(k+1)^{st}$ row with c_k, $-c_k$ on the diagonal. This becomes an orthogonal matrix if one selects $s_k^2 + c_k^2 = 1$. Note that

$$\overline{QB} = \begin{bmatrix} \rho_1 & \theta_2 & & & & & \\ & \rho_2 & \theta_3 & & & & \\ & & \ddots & \ddots & & & \\ & & & \rho_{k-1} & \theta_k & & \\ & & & & c_k\alpha_k + s_k\beta_{k+1} & s_k\alpha_{k+1} & \\ & & & & s_k\alpha_k - c_k\beta_{k+1} & -c_k\alpha_{k+1} & \\ & & & & & & \alpha_n \\ & & & & & & \beta_n \end{bmatrix} . \qquad \text{(B.23)}$$

At this point one sets $s_k\alpha_k - c_k\beta_{k+1} = 0$ and selects

$$s_k = \frac{\beta_{k+1}}{\sqrt{\alpha_k^2 + \beta_{k+1}^2}}, \quad c_k = \frac{\alpha_k}{\sqrt{\alpha_k^2 + \beta_{k+1}^2}}. \qquad \text{(B.24)}$$

This selection satisfies the previous criterion and then one can set

$$\begin{aligned} \rho_k &= c_k\alpha_k + s_k\beta_{k+1} = \left(\alpha_k^2 + \beta_{k+1}^2\right)^{1/2} \\ \theta_k &= s_k\alpha_{k+1} \\ \rho_{k+1} &= -c_k\alpha_{k+1} \end{aligned} \qquad \text{(B.25)}$$

Note that in the last step β_{n+1} is transformed to zero. This says that at the kth stage the rotation essentially performs

$$\begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} \begin{bmatrix} \bar{\rho}_k & 0 & \bar{\phi}_k \\ \beta_{k+1} & \alpha_{k+1} & 0 \end{bmatrix} = \begin{bmatrix} \rho_k & \theta_{k+1} & \phi_k \\ 0 & \bar{\rho}_{k+1} & \bar{\phi}_{k+1} \end{bmatrix}. \quad \text{(B.26)}$$

Now applying the matrix Q_k to the augmented matrix $\begin{bmatrix} B_k & \beta_1 e_1 \end{bmatrix}$ gives

$$Q_k \begin{bmatrix} B_k & \beta_1 e_1 \end{bmatrix} = \begin{bmatrix} Q_k B_k & Q_k \beta_1 e_1 \end{bmatrix}$$

$$= \begin{bmatrix} R_k & \varphi_k \\ 0 & \bar{\phi}_{k+1} \end{bmatrix} \qquad \text{(B.27)}$$

$$= \begin{bmatrix} \rho_1 & \theta_2 & 0 & \cdots & 0 & \phi_1 \\ 0 & \rho_2 & \theta_3 & \cdots & 0 & \phi_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \rho_{k-1} & \theta_k & \phi_{k-1} \\ \vdots & \vdots & \vdots & 0 & \rho_k & \phi_k \\ 0 & \cdots & \cdots & \cdots & 0 & \bar{\phi}_{k+1} \end{bmatrix}$$

The significance of this decomposition, called a QR factorization, is that we can write

$$\left\| B_k y_k - \beta_1 e_1 \right\|^2 = \left\| Q_k B_k y_k - Q_k \beta_1 e_1 \right\|^2 = \left\| R_k y_k - \varphi_k \right\|^2 + \left\| \bar{\phi}_k \right\|^2. \quad \text{(B.28)}$$

Therefore the left member is minimized if one solves $R_k y_k = f_k$ and the minimum is equal to the last term, where

$$\varphi_k = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_k \end{pmatrix}. \quad \text{(B.29)}$$

Although this relationship will not be needed directly it is interesting to note the relationship between t_{k+1} and $\bar{\phi}_{k+1}$ is given by

$$t_{k+1} = Q_k^T \begin{pmatrix} 0 \\ \bar{\phi}_{k+1} \end{pmatrix}. \quad \text{(B.30)}$$

This relationship follows from the orthogonality of Q_k and

$$Q_k t_{k+1} = Q_k (\beta_1 e_1) - Q_k B_k y_k = \begin{pmatrix} \varphi_k \\ \bar{\phi}_{k+1} \end{pmatrix} - \begin{pmatrix} R_k y_k \\ 0 \end{pmatrix} = \begin{pmatrix} \varphi_k - R_k y_k \\ \bar{\phi}_{k+1} \end{pmatrix} = \begin{pmatrix} 0 \\ \bar{\phi}_{k+1} \end{pmatrix} \quad \text{(B.31)}$$

under the assumption that y_k has been computed from $R_k y_k = \varphi_k$.

The previous relations make it possible to compute f_k directly without computing y_k or t_{k+1}. In particular write

$$f_k = V_k R_k^{-1} \varphi_k. \quad \text{(B.32)}$$

Define

$$D_k = V_k R_k^{-1} = \begin{bmatrix} d_1 & d_2 & \cdots & d_k \end{bmatrix}. \quad \text{(B.33)}$$

The columns of D_k can be found directly from

$$R_k^T D_k^T = V_k^T \quad \text{(B.34)}$$

by forward substitution. The significance of using forward substitution is that the columns can be computed iteratively as needed and then discarded. The columns are computed as follows. Set

$$D_k^T = \begin{pmatrix} d_1^T \\ \vdots \\ d_k^T \end{pmatrix}$$

$$R_k^T = \begin{bmatrix} \rho_1 & 0 & & 0 \\ \theta_2 & \rho_2 & & \\ & \ddots & \ddots & \\ 0 & & \theta_k & \rho_k \end{bmatrix}. \quad \text{(B.35)}$$

$$V_k^T = \begin{pmatrix} v_1^T \\ \vdots \\ v_k^T \end{pmatrix}$$

Then

$$R_k^T D_k^T = \begin{bmatrix} \rho_1 & 0 & & 0 \\ \theta_2 & \rho_2 & & \\ & \ddots & \ddots & \\ 0 & & \theta_k & \rho_k \end{bmatrix} \begin{pmatrix} d_1^T \\ \vdots \\ d_k^T \end{pmatrix} = \begin{pmatrix} \rho_1 d_1^T \\ \theta_2 d_1^T + \rho_2 d_2^T \\ \vdots \\ \theta_k d_{k-1}^T + \rho_k d_k^T \end{pmatrix} = \begin{pmatrix} v_1^T \\ \vdots \\ v_k^T \end{pmatrix} \quad \text{(B.36)}$$

which implies, setting $d_0 = x_0 = 0$,

$$d_1 = \frac{1}{\rho_1} v_1$$

$$d_2 = \frac{1}{\rho_2}(v_2 - \theta_2 d_1)$$

$$\vdots$$

$$d_k = \frac{1}{\rho_k}(v_k - \theta_k d_{k-1})$$

(B.37)

and

$$f_k = D_k \varphi_k = \begin{bmatrix} d_1 & \cdots & d_k \end{bmatrix} \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_k \end{pmatrix} = \begin{bmatrix} d_1 & \cdots & d_{k-1} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_{k-1} \end{pmatrix} + \phi_k d_k = f_{k-1} + \phi_k d_k. \quad (B.38)$$

This implies that f_k can be computed in an iterative fashion. The actual algorithm of Paige and Saunders [7, 8] introduces a slightly different but equivalent iteration. In particular, write

$$\rho_k d_k = v_k - \frac{\theta_k}{\rho_{k-1}}(\rho_{k-1} d_{k-1}). \quad (B.39)$$

Then set $w_k = \rho_k d_k$ and write the two step iteration

$$w_k = v_k - \frac{\theta_k}{\rho_{k-1}} w_{k-1}$$

$$f_k = f_{k-1} + \frac{\phi_k}{\rho_k} w_k$$

(B.40)

The iterative algorithm of Paige and Saunders [7, 8] can now be formulated as follows.

Step 1: Initialization

$$\beta_1 = \|g\|$$

$$u_1 = \frac{1}{\beta_1} g$$

$$\alpha_1 = \|H^T u_1\|$$

$$v_1 = \frac{1}{\alpha_1} H^T u_1 . \quad \text{(B.41)}$$

$$w_1 = v_1$$

$$x_0 = 0$$

$$\bar{\phi}_1 = \beta_1$$

$$\bar{\rho}_1 = \alpha_1$$

Step 2: Execute the outer loop

For $i = 1,2,3, \ldots$ do steps 3 through 6.

Step 3: Get the current values of u and v.

$$\beta_{i+1} = \|Hv_i - \alpha_i u_i\|$$

$$u_{i+1} = \frac{1}{\beta_{i+1}} (Hv_i - \alpha_i u_i)$$

$$\alpha_{i+1} = \|H^T u_{i+1} - \beta_{i+1} v_i\| \qquad \text{(B.42)}$$

$$v_{i+1} = \frac{1}{\alpha_{i+1}} (H^T u_{i+1} - \beta_{i+1} v_i)$$

Step 4: Apply the Givens rotation

$$\rho_i = \left(\bar{\rho}_i^2 + \beta_{i+1}^2 \right)^{1/2}$$

$$c_i = \frac{\bar{\rho}_i}{\rho_i}$$

$$s_i = \frac{\beta_{i+1}}{\rho_i}$$

$$\theta_{i+1} = s_i \alpha_{i+1} \qquad \text{(B.43)}$$

$$\bar{\rho}_{i+1} = -c_i \alpha_{i+1}$$

$$\phi_i = c_i \bar{\phi}_i$$

$$\bar{\phi}_{i+1} = s_i \bar{\phi}_i$$

Step 5: Update f and w

$$f_t = f_{t-1} + \frac{\phi_t}{\rho_t} w_t$$

$$w_{t+1} = v_{t+1} - \frac{\theta_{t+1}}{\rho_t} w_t$$
(B.44)

Step 6: Test for convergence

If the algorithm converged exit, otherwise return to step 2.

To test for convergence Paige and Saunders [6, 7] estimate the following items

$$\|r_k\|, \quad \|H^T r_k\|, \quad \|f_k\|, \quad \|H\|, \quad cond(H). \quad (B.45)$$

To estimate $\|r_k\|$ one begins with

$$r_k = U_{k+1} t_{k+1} = U_{k+1} Q_k^T \begin{pmatrix} 0 \\ \bar{\phi}_{k+1} \end{pmatrix} = \bar{\phi}_{k+1} U_{k+1} Q_k^T e_{k+1}. \quad (B.46)$$

Since the norm of orthogonal matrices is one and the norm of e_{k+1} is one then

$$\|r_k\| = \bar{\phi}_{k+1}. \quad (B.47)$$

But $\bar{\phi}_{k+1}$ is computed by

$$\bar{\phi}_{k+1} = \beta_1 s_1 s_2 \cdots s_k \quad (B.48)$$

so that

$$\|r_k\| = \beta_1 s_1 s_2 \cdots s_k \quad (B.49)$$

and since s_t in magnitude is less than one the iterative product pushes the norm to zero.

In estimating $\|H^T r_k\|$ one is really estimating the difference between the right and left sides of the normal equation at x_k. Begin with

$$H^T r_k = \bar{\phi}_{k+1} H^T U_{k+1} Q_k^T e_{k+1} = \bar{\phi}_{k+1} \left[V_k B_k^T + \alpha_{k+1} V_{k+1} e_{k+1}^T \right] Q_k^T e_{k+1}$$

$$= \bar{\phi}_{k+1} \left[V_k B_k^T Q_k^T e_{k+1} + \alpha_{k+1} V_{k+1} \left(e_{k+1}^T Q_k^T e_{k+1} \right) \right]$$
(B.50)

Note that

$$B_k^T Q_k^T = [Q_k B_k]^T = [R_k^T \quad 0]. \quad (B.51)$$

Then

$$H^T r_k = \bar{\phi}_{k+1} V_k [R_k^T \quad 0] e_{k+1} + \bar{\phi}_{k+1} \alpha_{k+1} \left(e_{k+1}^T Q_k^T e_{k+1} \right) V_{k+1}. \quad (B.52)$$

But $[R_k^T \quad 0] e_{k+1} = 0$ since $[R_k^T \quad 0]$ has k rows and $k+1$ columns with the $(k+1)$-st column identically zero and the vector e_{k+1} has all zero elements except with a one in the $(k+1)$-st element. Also $e_{k+1}^T Q_k^T e_{k+1} = -c_k$ since

$$e_{k+1}^T Q_k^T e_{k+1} = [0 \cdots \quad 0 \quad 1] \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & c_k & s_k \\ & & & s_k & -c_k \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} = [0 \quad \cdots \quad s_k \quad -c_k] \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} = -c_k \quad (B.53)$$

so that

$$H^T r_k = -\left(\bar{\phi}_k \alpha_{k+1} c_k \right) V_k. \quad (B.54)$$

Therefore

$$\left\| H^T r_k \right\| = \bar{\phi}_{k+1} \alpha_{k+1} |c_k|. \quad (B.55)$$

As long as α_{k+1} does not grow too rapidly then this decreases towards zero.

APPENDIX C: Sample Programs

NIST Standard Software Disclaimer

This software was developed at the National Institute of Standards and Technology by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is not subject to copyright protection and is in the public domain. The individual programs are part of an experimental system. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. Users of the programs assume sole responsibility under Federal law for determining the appropriateness of its use in any particular application; for any conclusions drawn from the results of its use; and for any actions taken or not taken as a result of analyzes performed using these tools. We would appreciate acknowledgement if the software is used.

INTENT AND USE

The algorithms, procedures, and computer programs described in this report constitute a methodology for predicting some of the consequences resulting from LADAR scans. They have been compiled from the best knowledge and understanding currently available, but have important limitations that must be understood and considered by the user. The program is intended for use by persons competent in the field of image processing and with some familiarity with personal computers. It is intended as an aid in reconstructing images blurred by LADAR optics.

PROGRAMS

The following program scripts are given in this appendix:

1. In Section C1 the Matlab script INT_DIST.m is given. This script takes a file generated by a LADAR and first produces a histogram of intensity values. The histogram usually shows two dominant peaks. The leftmost one in general represents background intensities and the rightmost one in general represents the bar code intensities. The background is stripped from the file and the bar codes displayed. The user has the option of saving the displayed figures in Postscript, Encapsulated Postscript, JPEG, or TIFF formats.
2. In Section C2 the Matlab script GENERATE_BAR_DATA.m is given. This script produces nine files of simulated ground truth data. It produces files for 0.0254 m (1 in) bars, 0.0508 m (2 in) bars and 0.1016 m (4 in) bars at 10 m, 20 m and 40 m distance.

3. In Section C3 a FORTRAN 90 program, called CONVOLVE.F90, program implements a fast convolution algorithm on simulated ground truth bar code data. The objects of the program are to test the effect of various convolution filters on the ground truth data and produce blurred images of bar code data that can be compared to bar code images acquired by LADAR scans. The program assumes the existence of a square array ground truth data set,f, and a square array, h, representing the convolution filter. The program produces a blurred array by performing a finite convolution summation

4. In Section C4 a FORTRAN 90 program, called DECONVOLVE.F90, program reconstructs ground truth bar code images from blurred images obtained from LADAR scans. The program assumes the existence of a square array scanned data set,g, and a square array, h, representing the convolution filter. The program produces a best estimate array of ground truth by applying an iterative least squares algorithm to a finite convolution summation. It is an inverse problem.

5. In Section C5.2 a FORTRAN 90 program, called SPREAD_FUNCTIO.F90 constructs by a least squares algorithm with residual correction an estimate of the kernel matrix that generates a given blurred image from a known ground truth image.

C1. MATLAB Script to Isolate Bar Codes

```
%****************************************************
%INT_DIST - Separates Bar Code intensity data from background
%          intensity.
%****************************************************
%This file employs a simple segmentation approach in order
%to eliminate the background intensities and distances. This
%brings the bar code intensities and distance measurements to
%the foreground. The assumption behind the technique is that
%when the image intensities are histogrammed they will fall
%into a histogram form with two prominent peaks. The first
%peak represents the backgound intensities and the second
%represents the bar code intensities. The algorithm to
%eliminate the background is simply to select out the intensity
%values less than 200.
%
%The input file for this script is assumed to have been
%generated by a LADAR scanning instrument and provides
%x, y, z, and intensity values of object hits. The units
%of the coordinates are in meters. The intensity values are
%integers between 0 and 255. The positive x coordinate is
%directed from the scanner towards the target object,
%positive y is towards the right and positive z is down.
%The origin is the upper left data point of the scan.
%
%Input file format:
%
%The first lines are header information lines beginning
%with #. These lines are followed by individual lines
%of data points in four columns. The first column is x,
%the second column is y, the third z, and the fourth
%intensity. The input file extension must be .txt.
%
%Output file formats:
%
%There are three output file types available:
%
%1. The first file will have the same name as the input file
%   with extension .dat. This file will contain only the data
%   points with the header lines from the input stripped off.
%   This file is automatically produced.
%2. The second file is a file of data points of the measured
%   bar code data extracted from the input data file with the
%   background noise removed. This is also automatically
```

```
%    produced.
%3. The third type of file is optional. Each of the figures
%    produced has a menu button that allows the image to be
%    stored in either Postscript (black and white), Postscript
%    (color), Encapsulated Postscript (black and white),
%    Encapsulated Postscript (color), JPEG, or TIFF.
%
%Usage instructions:
%
%1. First change the working directory in Matlab
%    to the directory with the script and data files.
%2. The current m-file is interactive. The user will be asked
%    to enter the minimum intensity value between peaks
%    in the histogram displayed when asked.
%3. For titles to plots, underscores in file names
%    are changed to hyphens for display
%4. Each plot has a menu button to print the plot to a file
%    in postscript, JPEG or TIFF.
%
% Author:
%                David E. Gilsinn
%                Mathematical and Computational Sciences Division
%                National Institute of Standards and Technology
%                100 Bureau Drive, Stop 8910
%                Gaithersburg, MD  20899-8910
%                e-mail: dgilsinn@nist.gov
%
%***********************************************************************


%Enter input file with .txt extension and create .dat file


file_name = input('Enter file name (omit .txt):','s');
infile = strcat(file_name,'.txt');
s = strcat(file_name,'.dat');
fidin = fopen(infile,'r');
fidout = fopen(s,'w');


%Strip the Header lines from the input file

while feof(fidin) ==0
        line = fgetl(fidin);
        match = findstr(line,'#');
        num = length(match);
        if num == 0
```

```matlab
                    fprintf(fidout,'%s\n',line);
        end
end
fclose(fidin);
fclose(fidout);

%load the .dat file

load ( s);

%Matlab loads the data file into a matrix with the same name
%as the file, but if the file_name begins with a number then
%Matlab adds an X in front of it.

%Test whether the file starts with a number. If so then add X
%in front. In either case read the matrix into the array a for
%easier handling.

if isletter(s(1)) == 0
        X_file_name = strcat('X',file_name);
        a = eval(X_file_name);
else
        a = eval(file_name);
end

%pull out array columns as vectors

dist = a(:,1);
y = -a(:,2);
z = a(:,3);
intensity = a(:,4);


%Generate histogram of intensities to estimate
%where to cut the image to get rid of background
%"noise". Set up image output menu.

f3 = figure(1);
hist(intensity,100);
xlabel('Intensity Level');
ylabel('Frequency Count');
s3 = strcat(file_name,'_intensity_hist');
new_file_name_3 = strrep(file_name,'_','-');
title(new_file_name_3);
set(f3,'NumberTitle','off',...
        'Name',s3);
```

```
set(f3,'NumberTitle','off',...
        'Name',s3);
pr3 = uimenu(f3,...
        'Label','  Print File');
ps3 = uimenu(pr3,...
        'Label','Postscript, Black and White',...
        'CallBack','print(f3,"-dps",s3)');
psc3 = uimenu(pr3,...
        'Label','Postscript, Color',...
        'CallBack','print(f3,"-dpsc",s3)');
eps3 = uimenu(pr3,...
        'Label','Encapsulated Postscript',...
        'CallBack','print(f3,"-deps",s3)');
epsc3 = uimenu(pr3,...
        'Label','Encapsulated Postscript, Color',...
        'CallBack','print(f3,"-depsc",s3)');
jpg3 = uimenu(pr3,...
        'Label','JPEG',...
        'CallBack','print(f3,"-djpeg90",s3)');
tiff3 = uimenu(pr3,...
        'Label','TIFF',...
        'CallBack','print(f3,"-dtiff",s3)');

%Filter out intensities less than 200

cutval = 200;

%Find all intensity values greater than or equal to cutval

k=find(intensity >= cutval);
y1 = y(k);
z1 = z(k);
int1 = intensity(k);




%Generate the Bar Code intensity image. This image
%will likely not have a square base.

f1 = figure(2);
ylin = linspace(min(y1),max(y1),257);
zlin = linspace(min(z1),max(z1),257);
[Y,Z] = meshgrid(ylin,zlin);
INT = griddata(y1,z1,int1,Y,Z,'cubic');
mesh(Y,Z,INT);
xlabel('y');
```

```
ylabel('z');
zlabel('Intensity');
s1 = strcat(file_name,'_intensity');
%change underlines to hyphens
new_file_name_1 = strrep(file_name,'_','-');
title(new_file_name_1);
set(f1,'NumberTitle','off',...
        'Name',s1);
pr1 = uimenu(f1,...
        'Label','  Print File');
ps1 = uimenu(pr1,...
        'Label','Postscript, Black and White',...
        'CallBack','print(f1,"-dps",s1)');
psc1 = uimenu(pr1,...
        'Label','Postscript, Color',...
        'CallBack','print(f1,"-dpsc",s1)');
eps1 = uimenu(pr1,...
        'Label','Encapsulated Postscript',...
        'CallBack','print(f1,"-deps",s1)');
epsc1 = uimenu(pr1,...
        'Label','Encapsulated Postscript, Color',...
        'CallBack','print(f1,"-depsc",s1)');
jpg1 = uimenu(pr1,...
        'Label','JPEG',...
        'CallBack','print(f1,"-djpeg90",s1)');
tiff1 = uimenu(pr1,...
        'Label','TIFF',...
        'CallBack','print(f1,"-dtiff",s1)');



%Generate an output file with a square base.
%The background base will have the value cutval

[row,col] = size(INT);
nanarray = isnan(INT);
[ii,jj] = find(nanarray);
for i = 1:length(ii)
   INT(ii(i),jj(i)) = cutval;
end
outfile = strcat(file_name,'_cut.txt');
fid1 = fopen(outfile,'w');
for j = 1:col
   for i = 1:row
      fprintf(fid1,'%6.2f\n',INT(i,j));
   end
```

```
end
fclose(fid1);


%
%Plot the filled in cut intensity picture
%
f4 = figure(3);
mesh(Y,Z,INT);
xlabel('y');
ylabel('z');
zlabel('Intensity');
s4 = strcat(file_name,'_intensity_cut');
%change underlines to hyphens
new_file_name_4 = strrep(file_name,'_','-');
title(new_file_name_4);
set(f4,'NumberTitle','off',...
        'Name',s4);
pr4 = uimenu(f4,...
        'Label','  Print File');
ps4 = uimenu(pr4,...
        'Label','Postscript, Black and White',...
        'CallBack','print(f4,"-dps",s4)');
psc4 = uimenu(pr4,...
        'Label','Postscript, Color',...
        'CallBack','print(f4,"-dpsc",s4)');
eps4 = uimenu(pr4,...
        'Label','Encapsulated Postscript',...
        'CallBack','print(f4,"-deps",s4)');
epsc4 = uimenu(pr4,...
        'Label','Encapsulated Postscript, Color',...
        'CallBack','print(f4,"-depsc",s4)');
jpg4 = uimenu(pr4,...
        'Label','JPEG',...
        'CallBack','print(f4,"-djpeg90",s4)');
tiff4 = uimenu(pr4,...
        'Label','TIFF',...
        'CallBack','print(f4,"-dtiff",s4)');

%
%expand the cut intensity figure
%

f5 = figure(4);
k = find(intensity > cutval);
new_intensity = intensity(k);
```

```matlab
new_y = y(k);
new_z = z(k);
new_ylin = linspace(min(new_y),max(new_y),257);
new_zlin = linspace(min(new_z),max(new_z),257);
[NEW_Y,NEW_Z] = meshgrid(new_ylin,new_zlin);
NEW_INT = griddata(new_y,new_z,new_intensity,NEW_Y,NEW_Z,'cubic');
mesh(NEW_Y,NEW_Z,NEW_INT);
xlabel('y');
ylabel('z');
zlabel('Intensity');
s5 = strcat(file_name,'_intensity_crop');
%change underlines to hyphens
new_file_name_5 = strrep(file_name,'_','-');
title(new_file_name_5);
set(f5,'NumberTitle','off',...
        'Name',s5);
pr5 = uimenu(f5,...
        'Label','  Print File');
ps5 = uimenu(pr5,...
        'Label','Postscript, Black and White',...
        'CallBack','print(f5,"-dps",s5)');
psc5 = uimenu(pr5,...
        'Label','Postscript, Color',...
        'CallBack','print(f5,"-dpsc",s5)');
eps5 = uimenu(pr5,...
        'Label','Encapsulated Postscript',...
        'CallBack','print(f5,"-deps",s5)');
epsc5 = uimenu(pr5,...
        'Label','Encapsulated Postscript, Color',...
        'CallBack','print(f5,"-depsc",s5)');
jpg5 = uimenu(pr5,...
        'Label','JPEG',...
        'CallBack','print(f5,"-djpeg90",s5)');
tiff5 = uimenu(pr5,...
        'Label','TIFF',...
        'CallBack','print(f5,"-dtiff",s5)');
```

C2. MATLAB Script to Create Simulated Bar Codes

```
%***********************************************************************
%GENERATE_BAR_DATA- This script produces ground truth files for
%                   the three bar code test configurations used to
%                   acquire measured bar code intensity data.
%***********************************************************************
% This file generates simulated ground truth image data sets in vector
% form of an approximately 1 meter by 1 meter surface with 9 bar codes
% placed in
% three rows with three bar codes each. This simulation attempts to
% approximately recreate an experiment in which a board with similar
% bar codes of reflector tape was probed by a LADAR beam at various
% distances. The bar codes were 6 in high and 4, 2, 1 in in width for
% the different trials. The 3 top bars were place 4 in apart, the three
% middle bars were placed 3 in below the top row and 2 in apart, while
% the lowest row was placed 3 in below the second with bars place 1 in
% apart. From experimental data the intensity of the background board
% is about 150 and the reflector tape intensity was about 250. This was
% on a range of 0 to 255.
%
% The size of the simulated ground truth image depends on the desired
% filter image size and the filter matrix size. In particular if the
% desired filter matrix is ma x ma and the desired filtered image is ng
% x ng elements then the ground truth image will be set to nf x nf
% where nf = ng + ma - 1. Thus, for a filtered image of side ng
% elements the unit step vertically is dx = 1/(ng-1) and the unit step
% horizontally is dy = 1/(ng-1). In order to make things somewhat easy
% to work with we assume that ng and ma are odd integers which
% also makes nf odd. The ground truth image will be ((ma-1)/2)*dx (or
% dy) units on a side greater than the filtered image.
%
%                 * * * * * * * * * * * * * * * * * * * * * * * * *
%                 *                                               *
%                 *  . . . . . . . . . . . . . . . . . . . . . .  *
%                 *  .                                         .  *
%                 *  .    +++     +++     +++      .  *
%                 *  .    +  +     +  +     +  +      .  *
%                 *  .    +++     +++     +++      .  *
%                 *  .                                   .  *
%                 *  .    +++     +++     +++      .  *
%                 *  .    +  +     +  +     +  +      .  *
%                 *  .    +++     +++     +++      .  *
%                 *  .                                   .  *
```

89

```
%              *  .  +++     +++     +++   .  *
%              *  .  + +     + +     + +   .  *
%              *  .  +++     +++     +++   .  *
%              *  .                     .  *
%              *  ...........................  *
%              *                              *
%              * * * * * * * * * * * * * * * * * * * * * * * * * *
%
% The figure above is a schematic of a board that holds the bar codes.
% The  outside boundary depicted by *** represents the ground truth
% image. Note that it is larger than the filtered image depicted by ...
% since the filter matrix at the boundary of the filtered emage
% overlaps the edge. Let f(i,j) represent the ground truth image. Then
% f(1,1) is taken in the upper left corner and f(nf,nf) is in the lower right. Grid spacing
% will be taken based on dx and dy defined above.
%
% The object of this script is to make nine files of barcode data for
% 4, 2, 1 in bars. These files will simulate ground truth images for
% each of three distances thus nine files. Distances are 10, 20 40
% meters. The files will be used to produce filtered images of size ng
% = 257 points on a side. Interval between  points will be
% approximately 0.004 meters (4 mm). At 10 m the filter will be
% ma = 15 points on a side, at 20 m the filter will be 17 points on a
% side and at 40 m it will be 29 points on a side. The number of points
% on a side of the ground truth image is determined by nf = ng + ma -1.
% The matrix structure of the file is: f(1,1) is in the upper left
% corner and f(nf,nf) is in the lower right. This is based on a beam of
% ma x ma points.
%
% x direction is down and y to the right (not LADAR coordinates)
% grid spacing in both x and y = 1/256 = 0.0039, set to 0.004
% bar data set has dimension 1 + ((ma-1)/2)*0.004 in both directions
% Extra amount beyond 1 m square is to accommodate the blurring operator
% at the edges.
%
% The files will be written out as vectors, one entry per line, in the
% form:
%        f(1,1), ..., f(nf,1),f(1,2),...,f(nf,2), ...,
%                          f(1,nf),...,f(nf,nf)
%
% Author:
%                David E. Gilsinn
%                Mathematical and Computational Sciences Division
%                National Institute of Standards and Technology
%                100 Bureau Drive, Stop 8910
%                Gaithersburg, MD  20899-8910
```

```
%                   e-mail: dgilsinn@nist.gov
%
%*****************************************************************

%
% Define output file names
%
% The Barfiles will contain (x, y, z) values of the simulated ground truth data
%
Barfiles = ['bars10m1in.txt';'bars10m2in.txt';'bars10m4in.txt';...
                  'bars20m1in.txt';'bars20m2in.txt';'bars20m4in.txt';...
                  'bars40m1in.txt';'bars40m2in.txt';'bars40m4in.txt'];
%
% The Intfiles will contain only the z values for plotting of the intensities in
% MATLAB
%
Intfiles = ['int10m1in.txt';'int10m2in.txt';'int10m4in.txt';...
                 'int20m1in.txt';'int20m2in.txt';'int20m4in.txt';...
                 'int40m1in.txt';'int40m2in.txt';'int40m4in.txt'];

Folder = 'c:\Bar_Code_data\';
Barcells = cellstr(Barfiles);
Intcells = cellstr(Intfiles);

%
% Define number of beam grid points per side
%

ma(1) = 15;
ma(2) = 17;
ma(3) = 29;

%
% Define the number of grid points per side for distorted image
%

ng = 257;

%
% Define grid spacing in meters
%

delta = 0.004;

%
%Specify bar code width (numbers refer to inches)
```

```
%

    bar_width(1) = 1;
    bar_width(2) = 2;
    bar_width(3) = 4;

%
% Generate the nine ground truth image vector files
% by stepping through each of the distances (10, 20, 40 m) and
% then through the bar code sizes (4, 2, 1)

%
% Step through each beam model for a distance
%

for i = 1:3

        %
        % Compute size of ground truth image on a side
        %

        nf = ma(i) + ng -1;

        %
        % Generate the x (downward) location of the
        % corners of the bar codes. Values are in meters
        % Bars are numbered as follows:
        % First index is the bar number
        % Lowest row, left to right: 1,2,3
        % Middle row: 4,5,6
        % Top row: 7,8,9
        % Second index is corner number on bar (clockwise):
        % Corner 1 is lower left, Corner 2 is upper left,
        % Corner 3 is upper right, Corner 4 is lower right.
        %
        % Offset downwards by the ground truth overlap
        %

        bar_x(7,2) = 0.1952 + ((ma(i)-1)/2)*delta;
        bar_x(7,3) = bar_x(7,2);
        bar_x(8,2) = bar_x(7,2);
        bar_x(8,3) = bar_x(7,2);
        bar_x(9,2) = bar_x(7,2);
        bar_x(9,3) = bar_x(7,2);
        bar_x(7,1) = bar_x(7,2) + 0.1524;
        bar_x(7,4) = bar_x(7,1);
```

```
bar_x(8,1) = bar_x(7,1);
bar_x(8,4) = bar_x(7,1);
bar_x(9,1) = bar_x(7,1);
bar_x(9,4) = bar_x(7,1);
bar_x(4,2) = bar_x(7,1) + 0.0762;
bar_x(4,3) = bar_x(4,2);
bar_x(5,2) = bar_x(4,2);
bar_x(5,3) = bar_x(4,2);
bar_x(6,2) = bar_x(4,2);
bar_x(6,3) = bar_x(4,2);
bar_x(4,1) = bar_x(4,2) + 0.1524;
bar_x(4,4) = bar_x(4,1);
bar_x(5,1) = bar_x(4,1);
bar_x(5,4) = bar_x(4,1);
bar_x(6,1) = bar_x(4,1);
bar_x(6,4) = bar_x(4,1);
bar_x(1,2) = bar_x(4,1) + 0.0762;
bar_x(1,3) = bar_x(1,2);
bar_x(2,2) = bar_x(1,2);
bar_x(2,3) = bar_x(1,2);
bar_x(3,2) = bar_x(1,2);
bar_x(3,3) = bar_x(1,2);
bar_x(1,1) = bar_x(1,2) + 0.1524;
bar_x(1,4) = bar_x(1,1);
bar_x(2,1) = bar_x(1,1);
bar_x(2,4) = bar_x(1,1);
bar_x(3,1) = bar_x(1,1);
bar_x(3,4) = bar_x(1,1);

%
%

for bar_w = 1:3

%
%
%

        %Define the bars y coordinates. bar_width is 4, 2 or 1
        %
        %First index is the bar number
        %Lowest row, left to right: 1,2,3
        %Middle row: 4,5,6
        %Top row: 7,8,9
        %
        %Entering y coordinates that change with
        %bar width working from the board center
        % 4 in = 0.1016 m, 2 in = 0.0508 m
```

```
% 1 in = 0.0254 m, 1/2 in = 0.0127
%

switch bar_width(bar_w)

        %
        % The case numbers are the bar width values
        %
        case 4
        %
        %Lowest row: row 1
        %

        bar_y(2,1) = ((1+(ma(i)-1)*delta)/2) - 0.0508;
        bar_y(2,2) = bar_y(2,1);
        bar_y(1,4) = bar_y(2,1) - 0.0254;
        bar_y(1,3) = bar_y(1,4);
        bar_y(1,1) = bar_y(1,4) - 0.1016;
        bar_y(1,2) = bar_y(1,1);
        bar_y(2,4) = ((1+(ma(i)-1)*delta)/2) + 0.0508;
        bar_y(2,3) = bar_y(2,4);
        bar_y(3,1) = bar_y(2,4) + 0.0254;
        bar_y(3,2) = bar_y(3,1);
        bar_y(3,4) = bar_y(3,1) + 0.1016;
        bar_y(3,3) = bar_y(3,4);


        %
        %Middle row: row 2
        %

        bar_y(5,1) = ((1+(ma(i)-1)*delta)/2) - 0.0508;
        bar_y(5,2) = bar_y(5,1);
        bar_y(4,4) = bar_y(5,1) - 0.0508;
        bar_y(4,3) = bar_y(4,4);
        bar_y(4,1) = bar_y(4,3) - 0.1016;
        bar_y(4,2) = bar_y(4,1);
        bar_y(5,4) = ((1+(ma(i)-1)*delta)/2) + 0.0508;
        bar_y(5,3) = bar_y(5,4);
        bar_y(6,1) = bar_y(5,4) + 0.0508;
        bar_y(6,2) = bar_y(6,1);
        bar_y(6,4) = bar_y(6,1) + 0.1016;
        bar_y(6,3) = bar_y(6,4);


        %
        %Top row: row 3
        %
```

```
bar_y(8,1) = ((1+(ma(i)-1)*delta)/2) - 0.0508;
bar_y(8,2) = bar_y(8,1);
bar_y(7,4) = bar_y(8,1) - 0.1016;
bar_y(7,3) = bar_y(7,4);
bar_y(7,1) = bar_y(7,4) - 0.1016;
bar_y(7,2) = bar_y(7,1);
bar_y(8,4) = ((1+(ma(i)-1)*delta)/2) + 0.0508;
bar_y(8,3) = bar_y(8,4);
bar_y(9,1) = bar_y(8,4) + 0.1016;
bar_y(9,2) = bar_y(9,1);
bar_y(9,4) = bar_y(9,1) + 0.1016;
bar_y(9,3) = bar_y(9,4);

%
%Write out data file
%
% Define the output file for each case
%
% Create the file names for the eighteen output files
%  Nine (x,y,z) files and nine z files
% output is the character string for an (x,y,z) data file
% output1 is the character sting for a z data file
%
file = (i-1)*3 + bar_w;
output = [Folder Barcells{file}];
output1 = [Folder Intcells{file}];
fid = fopen(output,'w');
fid1 = fopen(output1,'w');
for j = 1:nf
        y = (j-1)*delta;
        for k = 1:nf
                %go down rows first then colums
                x = (k-1)*delta;
                if ((bar_x(7,2) <= x) & ...
                                    (x <= bar_x(7,1)))...
                        & ( ((bar_y(7,2) <= y) & ...
                                    (y <= bar_y(7,3)))...
                            | ((bar_y(8,2) <= y) & ...
                                    (y <= bar_y(8,3)))...
                            | ((bar_y(9,2) <= y) & ...
                                    (y <= bar_y(9,3))) )
                    z = 250;
                elseif ((bar_x(4,2) <= x) & ...
                                    (x <= bar_x(4,1)))...
                        & ( ((bar_y(4,2) <= y) & ...
```

```matlab
                                        (y <= bar_y(4,3)))...
                               | ((bar_y(5,2) <= y) & ...
                                        (y <= bar_y(5,3)))...
                               | ((bar_y(6,2) <= y) & ...
                                        (y <= bar_y(6,3))) )
                        z = 250;
                elseif ((bar_x(1,2) <= x) & ...
                                        (x <= bar_x(1,1)))...
                        & ( ((bar_y(1,2) <= y) & ...
                                        (y <= bar_y(1,3)))...
                               | ((bar_y(2,2) <= y) & ...
                                        (y <= bar_y(2,3)))...
                               | ((bar_y(3,2) <= y) & ...
                                        (y <= bar_y(3,3))) )
                        z = 250;
                else
                        z = 150;
                end
                fprintf(fid,'%8.4f %8.4f %6.2f\n',...
                                                    x,y,z);
                fprintf(fid1,'%6.2f\n',z);
        end
end
fclose(fid);
fclose(fid1);

%
case 2
%
%Lowest row: row 1
%

bar_y(2,1) = ((1+(ma(i)-1)*delta)/2) - 0.0254;
bar_y(2,2) = bar_y(2,1);
bar_y(1,4) = bar_y(2,1) - 0.0254;
bar_y(1,3) = bar_y(1,4);
bar_y(1,1) = bar_y(1,4) - 0.0508;
bar_y(1,2) = bar_y(1,1);
bar_y(2,4) = ((1+(ma(i)-1)*delta)/2) + 0.0254;
bar_y(2,3) = bar_y(2,4);
bar_y(3,1) = bar_y(2,4) + 0.0254;
bar_y(3,2) = bar_y(3,1);
bar_y(3,4) = bar_y(3,1) + 0.0508;
bar_y(3,3) = bar_y(3,4);

%
```

```matlab
%Middle row: row 2
%

bar_y(5,1) = ((1+(ma(i)-1)*delta)/2) - 0.0254;
bar_y(5,2) = bar_y(5,1);
bar_y(4,4) = bar_y(5,1) - 0.0508;
bar_y(4,3) = bar_y(4,4);
bar_y(4,1) = bar_y(4,3) - 0.0508;
bar_y(4,2) = bar_y(4,1);
bar_y(5,4) = ((1+(ma(i)-1)*delta)/2) + 0.0254;
bar_y(5,3) = bar_y(5,4);
bar_y(6,1) = bar_y(5,4) + 0.0508;
bar_y(6,2) = bar_y(6,1);
bar_y(6,4) = bar_y(6,1) + 0.0508;
bar_y(6,3) = bar_y(6,4);


%
%Top row: row 3
%

bar_y(8,1) = ((1+(ma(i)-1)*delta)/2) - 0.0254;
bar_y(8,2) = bar_y(8,1);
bar_y(7,4) = bar_y(8,1) - 0.1016;
bar_y(7,3) = bar_y(7,4);
bar_y(7,1) = bar_y(7,4) - 0.0508;
bar_y(7,2) = bar_y(7,1);
bar_y(8,4) = ((1+(ma(i)-1)*delta)/2) + 0.0254;
bar_y(8,3) = bar_y(8,4);
bar_y(9,1) = bar_y(8,4) + 0.1016;
bar_y(9,2) = bar_y(9,1);
bar_y(9,4) = bar_y(9,1) + 0.0508;
bar_y(9,3) = bar_y(9,4);


%
%Write out data file
%
% Define the output file for each case
%

file = (i-1)*3 + bar_w;
output = [Folder Barcells{file}];
output1 = [Folder Intcells{file}];
fid = fopen(output,'w');
fid1 = fopen(output1,'w');
%
for j = 1:nf
```

```
            y = (j-1)*delta;
            for k = 1:nf
                    %go down rows first then colums
                    x = (k-1)*delta;
                    if ((bar_x(7,2) <= x) & ...
                                        (x <= bar_x(7,1)))...
                            & ( ((bar_y(7,2) <= y) & ...
                                        (y <= bar_y(7,3)))...
                                | ((bar_y(8,2) <= y) & ...
                                        (y <= bar_y(8,3)))...
                                | ((bar_y(9,2) <= y) & ...
                                        (y <= bar_y(9,3))) )
                            z = 250;
                    elseif ((bar_x(4,2) <= x) & ...
                                        (x <= bar_x(4,1)))...
                            & ( ((bar_y(4,2) <= y) & ...
                                        (y <= bar_y(4,3)))...
                                | ((bar_y(5,2) <= y) & ...
                                        (y <= bar_y(5,3)))...
                                | ((bar_y(6,2) <= y) & ...
                                        (y <= bar_y(6,3))) )
                            z = 250;
                    elseif ((bar_x(1,2) <= x) & ...
                                        (x <= bar_x(1,1)))...
                            & ( ((bar_y(1,2) <= y) & ...
                                        (y <= bar_y(1,3)))...
                                | ((bar_y(2,2) <= y) & ...
                                        (y <= bar_y(2,3)))...
                                | ((bar_y(3,2) <= y) & ...
                                        (y <= bar_y(3,3))) )
                            z = 250;
                    else
                            z = 150;
                    end
                    fprintf(fid,'%8.4f %8.4f %6.2f\n',...
                                                    x,y,z);
                    fprintf(fid1,'%6.2f\n',z);
            end
    end
    fclose(fid);
    fclose(fid1);

%
case 1
%
%Lowest row: row 1
```

```
%

bar_y(2,1) = ((1+(ma(i)-1)*delta)/2) - 0.0127;
bar_y(2,2) = bar_y(2,1);
bar_y(1,4) = bar_y(2,1) - 0.0254;
bar_y(1,3) = bar_y(1,4);
bar_y(1,1) = bar_y(1,4) - 0.0254;
bar_y(1,2) = bar_y(1,1);
bar_y(2,4) = ((1+(ma(i)-1)*delta)/2) + 0.0127;
bar_y(2,3) = bar_y(2,4);
bar_y(3,1) = bar_y(2,4) + 0.0254;
bar_y(3,2) = bar_y(3,1);
bar_y(3,4) = bar_y(3,1) + 0.0254;
bar_y(3,3) = bar_y(3,4);

%
%Middle row: row 2
%

bar_y(5,1) = ((1+(ma(i)-1)*delta)/2) - 0.0127;
bar_y(5,2) = bar_y(5,1);
bar_y(4,4) = bar_y(5,1) - 0.0508;
bar_y(4,3) = bar_y(4,4);
bar_y(4,1) = bar_y(4,3) - 0.0254;
bar_y(4,2) = bar_y(4,1);
bar_y(5,4) = ((1+(ma(i)-1)*delta)/2) + 0.0127;
bar_y(5,3) = bar_y(5,4);
bar_y(6,1) = bar_y(5,4) + 0.0508;
bar_y(6,2) = bar_y(6,1);
bar_y(6,4) = bar_y(6,1) + 0.0254;
bar_y(6,3) = bar_y(6,4);

%
%Top row: row 3
%

bar_y(8,1) = ((1+(ma(i)-1)*delta)/2) - 0.0127;
bar_y(8,2) = bar_y(8,1);
bar_y(7,4) = bar_y(8,1) - 0.1016;
bar_y(7,3) = bar_y(7,4);
bar_y(7,1) = bar_y(7,4) - 0.0254;
bar_y(7,2) = bar_y(7,1);
bar_y(8,4) = ((1+(ma(i)-1)*delta)/2) + 0.0127;
bar_y(8,3) = bar_y(8,4);
bar_y(9,1) = bar_y(8,4) + 0.1016;
bar_y(9,2) = bar_y(9,1);
```

```
bar_y(9,4) = bar_y(9,1) + 0.0254;
bar_y(9,3) = bar_y(9,4);

%
%Write out data file
%
% Define the output file for each case
%

file = (i-1)*3 + bar_w;
output = [Folder Barcells{file}];
output1 = [Folder Intcells{file}];
fid = fopen(output,'w');
fid1 = fopen(output1,'w');
%
for j = 1:nf
        y = (j-1)*delta;
        for k = 1:nf
                %go down rows first then colums
                x = (k-1)*delta;
                if ((bar_x(7,2) <= x) & ...
                                        (x <= bar_x(7,1)))...
                        & ( ((bar_y(7,2) <= y) & ...
                                        (y <= bar_y(7,3)))...
                        | ((bar_y(8,2) <= y) & ...
                                        (y <= bar_y(8,3)))...
                        | ((bar_y(9,2) <= y) & ...
                                        (y <= bar_y(9,3))) )
                        z = 250;
                elseif ((bar_x(4,2) <= x) & ...
                                        (x <= bar_x(4,1)))...
                        & ( ((bar_y(4,2) <= y) & ...
                                        (y <= bar_y(4,3)))...
                        | ((bar_y(5,2) <= y) & ...
                                        (y <= bar_y(5,3)))...
                        | ((bar_y(6,2) <= y) & ...
                                        (y <= bar_y(6,3))) )
                        z = 250;
                elseif ((bar_x(1,2) <= x) & ...
                                        (x <= bar_x(1,1)))...
                        & ( ((bar_y(1,2) <= y) & ...
                                        (y <= bar_y(1,3)))...
                        | ((bar_y(2,2) <= y) & ...
                                        (y <= bar_y(2,3)))...
                        | ((bar_y(3,2) <= y) & ...
                                        (y <= bar_y(3,3))) )
```

```matlab
                z = 250;
            else
                z = 150;
            end
            fprintf(fid,'%8.4f %8.4f %6.2f\n',...
                                                x,y,z);
            fprintf(fid1,'%6.2f\n',z);
        end
    end
    fclose(fid);
    fclose(fid1);
    %end switch on bar-width
    end
%end of bar_w cases
end
% end of distance cases
end
```

z = 250;

C3. FORTRAN 90 Convolution Program

C3.1 Program CONVOLVE.F90

```
!*****************************************************************
!
!CONVOLVE.F90
!
! This program implements a fast convolution algorithm on simulated
! ground truth bar code data. The objects of the program are to test
! the effect of various convolution filters on the ground truth
! data and produce blurred images of bar code data that can be
! compared to bar code images acquired by LADAR scans. The program
! assumes the existence of a square array ground truth data set,f, and
! a square array, h, representing the convolution filter. The
! program produces a blurred array by performing a finite convolution
! summation.
!
! Let
!       ma = side length of filter (integer),
!       nf = side length of ground truth image (integer),
!       ng = side length of filtered image (integer).
!
! These quantities are related by the formula
!
!                    nf = ng + ma - 1
!
! Note that this implies that ng < nf.
!
! The convolution filter is given by a matrix
!
!       h(ma,ma) = filter values
!
! The ground truth image is assumed to be given by a function
! defined at equally spaced points f(y(i),z(j)) where
!
!                    y(1) < y(2) < ... < y(nf)
!                    z(1) < z(2) < ... < z(nf)
!
! The resulting blurred image will be given by a function g(Y(p),Z(q))
! where
!
!                    Y(1) < Y(2) < ... < Y(ng)
!                    Z(1) < Z(2) < ... < Z(ng)
!
```

```
! The finite convolution summation is then given by
!
!                    q+ma-1  p+ma-1
!                    xxxxxx  xxxxx
!                      x       x
!    g(Y(p),Z(q)) =    x       x   h(Y(p)-y(i),Z(q)-z(j))*f(y(i),z(j))
!                      x       x
!                    xxxxxx  xxxxx
!                     j=q      i=p
!
! In the computation the functions f and g will be written as
! elongated vectors x and b and the convolution filter h will be
! represented in its matrix form.
!
! INPUT File Formats:
!
! 1. Parameter input file name is Convolve_Input.txt with format
!
! Line 1: ma, ng   Read as two integers (space 4dig space 4dig)
! Line 2: ma      Read as a character string, (4 characters)
! Line 3: Filter (Beam) matrix file name (Up to 120 character string)
! Line 4: Blurred image vector file name (Up to 120 character string)
! Line 5: Name of ground truth image (Up to 120 character string)
! Put a CR/LF (ENTER) at the end of line 5
!
! 2. Filter input file format:
!
! ma rows by ma columns. Each entry in G15.6.
!
! 3. Ground truth file format:
!
! Three columns in (F8.4,F9.4,F7.2). Column 1 is the y value, column 2
! is the z value, and column 3 is the image intensity value.
!
! Author:
!                    David E. Gilsinn
!                    Mathematical and Computational Sciences Division
!                    National Institute of Standards and Technology
!                    100 Bureau Drive, Stop 8910
!                    Gaithersburg, MD  20899-8910
!                    e-mail: dgilsinn@nist.gov
!
!******************************************************************
!
       PROGRAM Convolve
       IMPLICIT NONE
```

```fortran
!
!*****************************************************************
!
!
! The object of this program is to test the convolution algorithm
! on known bar code ground truth images.
!
! Parameter specifications:
!
        CHARACTER(*), PARAMETER :: Folder = "c:\Convolve\"
!
!
! Other varaible specifications:
!
! x is a vector representing the ground truth image stored by columns
! of length nf*nf. b is the filtered image stored by columns of length
! ng*ng.
!

        INTEGER ::  p, l, k, q ,e, ma, nf, ng
        INTEGER :: i, j, n, m, Inputstatus, Memorystatus, nout
        REAL, DIMENSION(:,:), ALLOCATABLE :: h
        REAL, DIMENSION(:), ALLOCATABLE ::  y1, z1, x, b
        REAL :: xx, yy,z, sum, residnorm, maxb, minb
        CHARACTER(120) :: BeamMatrix, GroundTruth, Beam, BeamL, GT, GTL,&
                          FilteredVect, Filtered, FilteredL, Output, &
                          OutputL,Printfile, Reconstruct, ReconstructL,&
                          PredGroundTruth, Input, InputL
        CHARACTER(4) :: CharMa, CharMaR
        CHARACTER(12) :: BeamFMT
!
! Get the parameter input file. Display input parameters.
!
        Input = Folder//"Convolve_Input.txt"
        InputL = ADJUSTL(Input)
        OPEN(UNIT = 2, FILE = InputL, IOSTAT = Inputstatus)
        IF (Inputstatus > 0)&
             STOP "*** Error on opening unit 2 Input file ***"
        READ(UNIT = 2, FMT = '(2I5)') ma, ng
        Print *,ma,ng
        READ(UNIT = 2, FMT = '(A4)') CharMa
        Print *,CharMa
        READ(UNIT = 2, FMT = '(A120)') Beam
        Print *, Beam
        READ(UNIT = 2, FMT = '(A120)') Filtered
        Print *, Filtered
```

```fortran
        READ(UNIT = 2, FMT = '(A120)') Reconstruct
        Print *, Reconstruct
        CLOSE(2)
        BeamL = ADJUSTL(Beam)
        FilteredL = ADJUSTL(Filtered)
        OutputL = ADJUSTL(Output)
        ReconstructL = ADJUSTL(Reconstruct)
        CharMaR = ADJUSTR(CharMa)
        BeamMatrix = Folder//BeamL
        print *,'BeamMatrix = ',BeamMatrix
        FilteredVect = Folder//FilteredL
        PredGroundTruth = Folder//ReconstructL
        BeamFMT = '('//CharMaR//'G15.6)'
!
! Allocate arrays
!
! Get lengths of ground truth image vector, n, and filtered image
! vector, m.
!
        nf = ng + ma -1
        n = nf*nf
        m = ng*ng
        ALLOCATE(h(ma,ma), y1(n), z1(n), x(n), b(m), STAT = Memorystatus)
        IF (Memorystatus /= 0) STOP "*** Memory allocation error ***"
!
! Get the filter data as a matrix.
!
        OPEN(UNIT=4,FILE = BeamMatrix,STATUS = 'OLD',IOSTAT = Inputstatus)
        IF (Inputstatus > 0 ) THEN
                PRINT *, BeamMatrix
                STOP "*** Error on opening unit 4 ***"
        END IF
        DO i = 1,ma
                READ(UNIT = 4, FMT = BeamFMT) (h(i,j), j = 1,ma)
        END DO
        CLOSE(4)
!
! Get the ground truth image image
!
        OPEN(UNIT = 6, FILE = PredGroundTruth,STATUS = 'OLD')
        i = 0
        DO
                READ(UNIT = 6,FMT = '(F8.4,F9.4,F7.2)',IOSTAT = Inputstatus)&
                                yy, z, xx
                IF (Inputstatus > 0) STOP "*** Input Error on Unit 6 ***"
                IF (Inputstatus < 0) EXIT ! End of file
```

```fortran
                    i = i + 1
                    y1(i) = yy ! y value of data points (Not used)
                    z1(i) = z  ! z value of data points (Not used)
                    x(i) = xx
           END DO
           CLOSE(6)
           Print *, 'gt image. Next integers should be equal.', i, n
!
! Perform the convolution algorithm
!
           DO p = 1,ng                    ! Loop over each y block of ng elements
                    DO l = 1,ng           ! Local row number within y block
                            j = (p-1)*ng + l
                            sum = 0.0
                            DO k = 0,ma-1          ! Loop over x blocks of nf elements
                                    DO q = 0,ma-1          ! Form the inner product for row j
                                            e = (p + k - 1)*nf +l + q
                                            sum = sum + h(ma-q,ma-k)*x(e)

                                    END DO
                            END DO
                            b(j) =  sum
                    END DO
           END DO
!
! Scale the output intensities to between 140 and 255
! Write the output as a vector for Matlab graphics
!
           minb = MINVAL(b)
           maxb = MAXVAL(b)

                    OPEN(UNIT = 8,FILE = FilteredVect, STATUS = 'UNKNOWN')
           DO i = 1,m
                    b(i) = ((255.0-140.0)/(maxb-minb))*(b(i)-maxb) + 255.0
                    WRITE (UNIT = 8, FMT = '(1X, G12.6)')  b(i)
           END DO
           CLOSE(8)
!
! Deallocate memory
!
           DEALLOCATE(h, y1, z1, x, b )
!

END PROGRAM Convolve
```

C3.2 Sample Input Parameter File

```
 11  095
 11
spot_quarterin_10m.txt
Quarterin_conv_Bar_1in_10m_95x95.txt
yzint_Bar_1in_10m_data_105x105.txt
```

C4. FORTRAN 90 Deconvolution Program

C4.1 Program DECONVOLVE.F90

```
!*************************************************************************
!
!DECONVOLVE.F90
!
! This program reconstructs ground truth bar code images from
! blurred images obtained from LADAR scans. The program
! assumes the existence of a square array scanned data set,g, and
! a square array, h, representing the convolution filter. The
! program produces a best estimate array  of ground truth
! by applying an iterative least squares algorithm to a
! finite convolution summation. It is an inverse problem.
!
! Let
!         ma = side length of filter (integer),
!         nf = side length of ground truth image (integer),
!         ng = side length of filtered image (integer).
!
! These quantities are related by the formula
!
!                         nf = ng + ma - 1
!
! Note that this implies that ng < nf.
!
! The convolution filter is given by a matrix
!
!         h(ma,ma) = filter values
!
! The ground truth image is assumed to be given by a function
! defined at equally spaced points f(y(i),z(j)) where
!
!                         y(1) < y(2) < ... < y(nf)
!                         z(1) < z(2) < ... < z(nf)
!
! This will be the function reconstructed.
!
! The scanned blurred image will be given by a function g(Y(p),Z(q))
! where
!
!                         Y(1) < Y(2) < ... < Y(ng)
!                         Z(1) < Z(2) < ... < Z(ng)
!
```

```
! The finite convolution summation is then given by
!
!                    q+ma-1  p+ma-1
!                    xxxxxx  xxxxxx
!                       x       x
!    g(Y(p),Z(q)) =  x          x   h(Y(p)-y(i),Z(q)-z(j))*f(y(i),z(j))
!                       x       x
!                    xxxxxx  xxxxxx
!                      j=q      i=p
!
! In the computation the functions f and g will be written as
! elongated vectors x and b and the convolution filter h will be
! represented in its matrix form. The object is to use the filter h
! and the scanned data g to obtain a best estimate of f.
!
! INPUT File Formats:
!
! 1. Parameter input file name is LSQR_Input.txt with format
!
! Line 1: ma, ng   Read as two integers, FORMAT (2I5)
!                              i.e. space 4dig space 4dig
! Line 2: ma       Read as a character string, FORMAT (A4)
! Line 3: Beam matrix file name  Character string FORMAT (A120)
! Line 4: Filtered image vector file name  Character string
!                       FORMAT (A120)  (file in F6.2)
! Line 5: Name of LSQR output file FORMAT (A120)
! Line 6: Name of predicted ground truth image FORMAT (A120)
! Line 7: damp - regularization parameter FORMAT (F5.3)
! Line 8: atol - relative error in data defining A matrix.
!                           For 3 figures 0.001. FORMAT (F8.6)
! Line 9: btol - relative error in data defining right hand b.
!                           For 3 figures 0.001. FORMAT (F8.6)
! Put a CR/LF (ENTER) at the end of line 9
!
!
! 2. Filter input file format:
!
! ma rows by ma columns. Each entry in G15.6.
!
! 3. Scanned data file format:
!
! Three columns in (F8.4,F9.4,F7.2). Column 1 is the y value, column 2
! is the z value, and column 3 is the image intensity value.
!
! Subroutines required:
!
```

```fortran
! 1. SUBROUTINE Aprod((Mode, m, n, x, y, Leniw, Lenrw, Iw, Rw)
!        This subroutine computes the following:
!                     If Mode = 1, set y = y + H*x
!                     If Mode = 2, set x = x + (H^T) * y
!                             where H^T is the transpose of H formed from h.
!        Leniw, Lenrw are the length of the working arrays Iw, Rw
!
! 2.SUBROUTINE LSQR( M,N,APROD,DAMP,
!    1            LENIW,LENRW,IW,RW,
!    2            U,V,W,X,SE,
!    3            ATOL,BTOL,CONLIM,ITNLIM,NOUT,
!    4            ISTOP,ANORM,ACOND,RNORM,ARNORM,XNORM)
!   This subroutine performs the iterative least squares algorithm.
!   For a description of the algorithm see
!        (a) Paige, C., Saunders, M, "LSQR: An algorithm for sparse linear
!          equations and sparse least squares", ACM Transactions on
!          Mathematical Software, Vol 8, No. 1, 1982, 43-71.
!   (b) Paige, C., Saunders, M, "Algorithm 583, LSQR: Sparse linear
!          equations and least squares problems", ACM Transactions on
!          Mathematical Software, Vol 8, No. 2, 1982, 195-209.
!
!   The subroutine can be obtained from the ACM web site at
!        http://www.acm.org/calgo/contents/. It is not included in
!   the current code publication for copyright reasons.
!
! Author:
!                     David E. Gilsinn
!                     Mathematical and Computational Sciences Division
!                     National Institute of Standards and Technology
!                     100 Bureau Drive, Stop 8910
!                     Gaithersburg, MD 20899-8910
!                     e-mail: dgilsinn@nist.gov
!
!*************************************************************************
        MODULE LSQR_global
!*************************************************************************
!
! This module provides access to the quantities ma, nf, ng, h from
! the subroutines.
!
!*************************************************************************
!
        INTEGER :: ma, nf, ng
        REAL, DIMENSION(:,:), ALLOCATABLE :: h
!
        END MODULE LSQR_global
```

```fortran
!
!***************************************************************
!
! The main program DECONVOLVE begins here
!
!***************************************************************
        PROGRAM Deconvolve
        USE LSQR_global
        IMPLICIT NONE
        EXTERNAL Aprod, LSQR
!***************************************************************
!
! Parameter specifications
!
        INTEGER, PARAMETER :: Leniw = 1, Lenrw = 1
        CHARACTER(*), PARAMETER :: Folder = "c:\Deconvolve\"
!
! Other specifications
!
! x is a vector representing the ground truth image stored by columns of
! length nf*nf. b is the filtered image stored by columns of length ng*ng.
!
        INTEGER, DIMENSION(Leniw) :: Iw
        INTEGER :: i, j, n, m, Mode, Inputstatus, Memorystatus, nout, istop, mout(16),&
                   itnlim
        INTEGER, DIMENSION(8) :: Val
        REAL, DIMENSION(Lenrw) :: Rw
        REAL :: xx, atol, btol, conlim, damp, anorm, acond, rnorm, arnorm, xnorm, y,z
        REAL, DIMENSION(:), ALLOCATABLE :: x, b, u, v, w, se
        CHARACTER(120) :: BeamMatrix, GroundTruth, Beam, BeamL, GT, GTL,&
                   Input, InputL,&
                   FilteredVect, Filtered, FilteredL, Output, OutputL,&
                   Printfile, Reconstruct, ReconstructL,PredGroundTruth
        CHARACTER(4) :: CharMa, CharMaR
        CHARACTER(12) :: BeamFMT
        CHARACTER(LEN=12) :: Real_clock(3)
!
! Read the parameter input file
!
        Input = Folder//"LSQR_Input.txt"
        InputL = ADJUSTL(Input)
        OPEN(UNIT = 2, FILE = InputL, IOSTAT = Inputstatus)
        IF (Inputstatus > 0) STOP "*** Error on opening unit 2 Input file ***"
        READ(UNIT = 2, FMT = '(2I5)') ma, ng
        Print *,ma,ng
        READ(UNIT = 2, FMT = '(A4)') CharMa
```

```fortran
      Print *,CharMa
      READ(UNIT = 2, FMT = '(A120)') Beam
      Print *, Beam
      READ(UNIT = 2, FMT = '(A120)') Filtered
      Print *, Filtered
      READ(UNIT = 2, FMT = '(A120)') Output
      Print *, Output
      READ(UNIT = 2, FMT = '(A120)') Reconstruct
      Print *, Reconstruct
      READ(UNIT = 2, FMT = '(F5.3)') damp
      Print *, damp
      READ(UNIT = 2, FMT = '(F8.6)') atol
      Print *, atol
      READ(UNIT = 2, FMT = '(F8.6)') btol
      Print *, btol
      CLOSE(2)
      BeamL = ADJUSTL(Beam)
      FilteredL = ADJUSTL(Filtered)
      OutputL = ADJUSTL(Output)
      ReconstructL = ADJUSTL(Reconstruct)
      CharMaR = ADJUSTR(CharMa)
      BeamMatrix = Folder//BeamL
      FilteredVect = Folder//FilteredL
      Printfile = Folder//OutputL
      PredGroundTruth = Folder//ReconstructL
      BeamFMT = '('//CharMaR//'F15.6)'
!
! Allocate arrays
!
! Get lengths of ground truth image vector, n, and filtered image vecctor, m.
!
      nf = ng + ma -1
      n = nf*nf
      m = ng*ng
      ALLOCATE(h(ma,ma), x(n), b(m), u(m), v(n), w(n), se(n), STAT =
Memorystatus)
      IF (Memorystatus /= 0) STOP "*** Memory allocation error ***"
!
! Get the filter data as a matrix.
!
      OPEN(UNIT = 4,FILE = BeamMatrix, STATUS = 'OLD', IOSTAT = Inputstatus)
      IF (Inputstatus > 0 ) THEN
            PRINT *, BeamMatrix
            STOP "*** Error on opening unit 4 ***"
      END IF
      DO i = 1,ma
```

```fortran
            READ(UNIT = 4, FMT = BeamFMT) (h(i,j), j = 1,ma)
            Print *, (h(i,j), j = 1,ma)
      END DO
      CLOSE(4)
!
! Get the scanned data image
!
      OPEN(UNIT = 6, FILE = FilteredVect,      STATUS = 'OLD')
      i = 0
      DO
            READ(UNIT = 6,FMT = '(F8.4,F9.4,F7.2)',&
                        IOSTAT = Inputstatus) y, z, xx
            IF (Inputstatus > 0) STOP "*** Input Error on Unit 6 ***"
            IF (Inputstatus < 0) EXIT ! End of file
            i = i + 1
            b(i) = xx
      END DO
      CLOSE(6)
      Print *, 'Filtered image', i, m
!
! Assign b to u for input to LSQR. Note u is overwritten.
!
      u = b
!
! Dummy values. Work arrays not used.
!
      Iw(1) = 0
      Rw(1) = 0.0
!
! Setting up parameters for LSQR. Open an internal file
! used by LSQR for diagnostic output and a file to save
! the reconstructed image, x. Time calls are made for
! diagnostic information.
!
!
      nout = 7
      OPEN(UNIT = nout, FILE = Printfile,       STATUS = 'UNKNOWN')
      OPEN(UNIT = 8, FILE = PredGroundTruth, STATUS = 'UNKNOWN')
      conlim = 1.0e+7
      itnlim = 10
      CALL DATE_AND_TIME(Real_clock(1), Real_clock(2),&
                              Real_clock(3), Val)
      Print *, Val(1), Val(2) , Val(3), Val(5), Val(6),&
                        Val(7), Val(8)
      Call LSQR( m,n,Aprod,damp,Leniw,Lenrw,Iw,Rw,u,v,w,x,se,&
                  atol,btol,conlim,itnlim,nout,istop,anorm,&
```

```fortran
                    acond,rnorm,arnorm,xnorm )
        CALL DATE_AND_TIME(Real_clock(1), Real_clock(2),&
                           Real_clock(3), Val)
        PRINT *, Val(1), Val(2) , Val(3), Val(5), Val(6),&
                           Val(7), Val(8)
        DO i = 1,n
              WRITE (UNIT = 8, FMT = '(1X, F12.6)') x(i)
        END DO
!
        CLOSE(nout)
        CLOSE(8)
!
! Deallocate memory
!
        DEALLOCATE(h, x, b, u, v, w, se )
!
!
END PROGRAM Deconvolve
!*****************************************************************
!
        SUBROUTINE Aprod (Mode, m, n, x, y, Leniw, Lenrw, Iw, Rw)

! This version of Aprod performs a convolution of a filter matrix, h,
! with an image  vector, x. It also applies the transpose of h to a
! filtered image vector y. The storage of images is by columns.
!
! The image vector is stored in x of length n. It is generated from
! a square image of side nf so that n = nf*nf. If f(i,j), i, j = 1,nf
! is the image, the structure of x is as follows:
! x(1) = f(1,1), x(2) = f(2,1), ..., x(nf) = f(nf, 1),
! x(nf + 1) = x(1, 2), ..., x(2*nf) = x(nf, 2),...,
! x(nf*nf) = f(nf, nf).
!
! The blurred image vector is stored in y of length m. It is generated
! from a square image of side ng so that m = ng*ng. If, g(i, j),
! i, j = 1, ng is the blurred image, the ! structure of y is as follows:
! y(1) = g(1, 1), y(2) = g(2, 1), ..., y(ng) = g(ng, 1),
! y(ng + 1) = g(1, 2), ..., y (2*ng) = ! g(ng, 2), ...,
! y(ng*ng) = g(ng, ng)
!
! The matrix A is the sparse block Toeplitz matrix formed from the
! filter matrix, h, of size ma x ma. In this subroutine the sparse
! matrix is never created. Outputs are created by rows using the filter
! matrix. This reduces storage requirements.
!
! Aprod performs the following function:
```

```
!
!          If Mode = 1, set y = y + A*x
!          If Mode = 2, set x = x + (A^T) * y
!
! Iw and Rw are work arrays of length Leniw and Lenrw respectively.
! They are not used in this version of Aprod
!
! Module LSQR_global contains h, ma, ng, nf
!
!*********************************************************************
          USE LSQR_global
!
!

          IMPLICIT NONE
          INTEGER :: n, m, Leniw, Lenrw, Mode
          INTEGER :: p, l, j, k, q, e
          REAL :: x(n), y(m), Iw(Leniw), Rw(Lenrw)
          REAL :: sum, yj
!
!
!
!
!
!*****************************************************************
!
!          Select the mode
!
!*****************************************************************
!
          IF (mode == 1) THEN
!
!*****************************************************************
!
!          Mode = 1  -- Set y = y + A*x
!
!*****************************************************************
!
          DO p = 1,ng               ! Loop over each y block of ng elements
                DO l = 1,ng         ! Local row number within y block
                      j = (p-1)*ng + l
                      sum = 0.0
                      DO k = 0,ma-1            ! Loop over x blocks of nf elements
                            DO q = 0,ma-1       ! Form the inner product for row j
                                  e = (p + k - 1)*nf +l + q
                                  sum = sum + h(ma-q,ma-k)*x(e)
```

```
                        END DO
                  END DO
                  y(j) = y(j) + sum        ! Add A*x to y for row j
            END DO
      END DO
!
      ELSE
!
!******************************************************************
!
!     Mode = 2 -- Set x = x + (A^T)*y
!
!******************************************************************
!
      DO p = 1,ng
            DO l = 1,ng
                  j = (p-1)*ng + l
                  yj = y(j)
                  DO k = 0,ma-1
                        DO q = 0,ma-1
                              e = (p+k-1)*nf + l + q
                              x(e) = x(e) + h(ma-q,ma-k)*yj
                        END DO
                  END DO
            END DO
      END DO
!
      END IF
!
      END SUBROUTINE Aprod
```

C4.2 Sample Input Parameter File

```
  11  095
11
Beam_10m_gaus_sig06_3x7.txt
yzint_a_2in_10m_crop_95.txt
LSQR_decon_out_2in_10m.txt
decon_gaussig06_3x7_a_2in_10m_gt.txt
0.001
0.000001
0.000001
```

C5. FORTRAN 90 Program to Reverse Engineer the Beam Spread Function

C5.1 Matrix alignment

In order to apply the LSQR algorithm to the reverse engineering problem the matrices involved need to be stored in such a way that they align properly. As in the convolution and deconvolution programs the scanned data, g, is stored by columns. That is

$$g = \begin{pmatrix} g(1,1) \\ g(2,1) \\ \vdots \\ g(ng,1) \\ g(1,2) \\ \vdots \\ g(ng,2) \\ \vdots \\ g(1,ng) \\ \vdots \\ g(ng,ng) \end{pmatrix} \qquad (C5.1)$$

Similarly the unknown filter matrix, H, is also stored in vector form as

$$H = \begin{pmatrix} H(1,1) \\ H(2,1) \\ \vdots \\ H(ma,1) \\ H(1,2) \\ \vdots \\ H(ma,2) \\ \vdots \\ H(1,ma) \\ \vdots \\ H(ma,ma) \end{pmatrix} \qquad (C5.2)$$

The ground truth image is also assumed to be stored in column form as

117

$$f = \begin{pmatrix} f(1) \\ f(2) \\ \vdots \\ f(nf) \\ f(nf+1) \\ \vdots \\ f(2nf) \\ \vdots \\ f((nf-1)nf) \\ \vdots \\ f(nf^2) \end{pmatrix} = \begin{pmatrix} F(1,1) \\ F(2,1) \\ \vdots \\ F(nf,1) \\ F(1,2) \\ \vdots \\ F(nf,2) \\ \vdots \\ F(nf,nf-1) \\ \vdots \\ F(nf,nf) \end{pmatrix} \qquad (C5.3)$$

In order to satisfy the requirements of LSQR, the ground truth image has to be stored as a matrix for use in a modified version of the Aprod subroutine. The ground truth image is stored in a matrix of size $A(ng^2, ma^2)$.

$$A = \begin{bmatrix} f(1) & \cdots & f(ma) & f(nf+1) & \cdots & f(nf+ma) & \cdots & f((ma-1)nf+1) & \cdots & f((ma-1)nf+ma) \\ f(2) & \cdots & f(ma+1) & f(nf+2) & \cdot & f(nf+ma+1) & \cdots & f((ma-1)nf+2) & \cdots & f((ma-1)nf+ma+1) \\ & \cdots & & & \cdots & & \cdots & & \vdots \\ f(ng) & \cdot & f(ng+(ma-1)) & f(nf+ng) & \cdots & f(nf+ng+(ma-1)) & \cdots & f((ma-1)nf+ng) & \cdots & f((ma-1)nf+ng+(ma-1)) \\ f(nf+1) & \cdots & f(nf+1+(ma-1)) & f(2nf+1) & \cdots & f(2nf+1+(ma-1)) & \cdots & f((ma)nf+1) & \cdots & f((ma)nf+1+(ma-1)) \\ \vdots & & \vdots & & & \vdots & & & \vdots \\ f(nf+ng) & \cdot & f(nf+ng+(ma-1)) & f(2nf+ng) & \cdots & f(2nf+ng+(ma-1)) & \cdot & f((ma)nf+ng) & \cdots & f((ma)nf+ng+(ma-1)) \\ f(2nf+1) & \cdot & f(2nf+1+(ma-1)) & f(3nf+1) & \cdot & f(3nf+1+(ma-1)) & \cdot & f((ma+1)nf+1) & \cdot & f((ma+1)nf+1+(ma-1)) \\ \vdots & \cdots & \vdots & & \cdot & \vdots & & \vdots & \cdots & \vdots \\ f(2nf+ng) & \cdots & f(2nf+ng+(ma-1)) & f(3nf+ng) & \cdot & f(3nf+ng+(ma-1)) & \cdots & f((ma+1)nf+ng) & \cdots & f((ma+1)nf+ng+(ma-1)) \\ \vdots & \cdots & \vdots & & \cdots & \vdots & & \vdots & \cdots & \vdots \\ f((ng-1)nf+1) & \cdots & f((ng-1)nf+1+(ma-1)) & f((ng)nf+1) & \cdots & f((ng)nf+1+(ma-1)) & \cdots & f((ng-1+ma-1)nf+1) & \cdot & f((ng-1+ma-1)nf+1+(ma-1)) \\ \vdots & \cdots & \vdots & & \cdots & \vdots & & \vdots & \cdots & \vdots \\ f((ng-1)nf+ng) & \cdots & f((ng-1)nf+ng+(ma-1)) & f((ng)nf+ng) & \cdots & f((ng)nf+ng+(ma-1)) & \cdots & f((ng-1+ma-1)nf+ng) & \cdots & f((ng-1+ma-1)nf+ng+(ma-1)) \end{bmatrix} \quad (C5.4)$$

The storage is accomplished in the main program by the following portion of code. The transpose, AT, is also computed.

```
DO cc = 0, ng-1
      DO dd = 0, ng-1
            DO ee = 0, ma-1
                  DO ff = 1,ma
                        k1 = cc*ng+dd+1
                        k2 = ee*ma+ff
                        k3 = (ee+cc)*nf + dd + ff
                        A(k1,k2) = F(k3)
                  END DO
            END DO
      END DO
END DO
DO cc = 1,m
      DO dd = 1,ma2
```

$$AT(dd,cc) = A(cc,dd)$$
        END DO
END DO

C5.2 SPREAD_FUNCTION.F90 program

```
!*************************************************************************
! SPREAD_FUNCTION.F90
!
! The object of this program is to construct by a least squares
! algorithm  with residual correction an estimate of the kernel
! matrix that generates a given blurred image from a known ground
! truth image.
!
! INPUT FILES:
!
! 1. Parameter input file format
!
! Line 1: ma, ng   Read as two integers, FORMAT 2I5 i.e. space
!                                 4dig space 4dig
! Line 2: Charma  Character form of ma
! Line 3: Ground Truth file name  Character string (file in F6.2)
! Line 4: maximum number of kernel update iterations (at least 1
!                                 is performed)
! Line 5: Filtered image vector file name  Character string
!                                 (file in F8.4,F9.4,F7.2)
! Line 6: Name of LSQR output file
! Line 7: Name of predicted spread function
! Line 8: damp - regularization parameter
! Line 9: atol - relative error in data defining A matrix.
!                                 For 3 figures 0.001.
! Line 10: btol - relative error in data defining right hand b.
!                                 For 3 figures 0.001
! Put a CR/LF (ENTER) at the end of line 10
!
! 2. Filtered or scanned file image
!
! The file is assumed to be in three columns in (F8.4,F9.4,F7.2).
! Column 1 is the y value, column 2 is the z value, and column 3
! is the image intensity value.
!
! 3, Ground truth image
!
! The file is assumed to be in three columns in (F8.4,F9.4,F7.2).
! Column 1 is the y value, column 2 is the z value, and column 3
```

```
! is the image intensity value.
!
! Subroutines Required: LSQR, Aprod, Maxnorm
!
! Author:
!               David E. Gilsinn
!               Mathematical and Computational Sciences Division
!               National Institute of Standards and Technology
!               100 Bureau Drive, Stop 8910
!               Gaithersburg, MD  20899-8910
!
!               e-mail: dgilsinn@nist.gov
!
!********************************************************************
!
! This module provides global access to ma, nf, ng, A, AT
!
! where
!
! ma = side length of filter,
! nf = side length of ground truth image
! ng = side length of filtered image, nf = ng + ma -1
! A(ng*ng,ma*ma) - Allocatable Ground truth as a matrix
! AT(ma*ma,ng*ng) - Allocatable transpose of ground truth
!
!********************************************************************
        MODULE LSQR_global
!********************************************************************
!
        INTEGER :: ma, nf, ng
        REAL, DIMENSION(:,:), ALLOCATABLE :: A, AT
!
        END MODULE LSQR_global
!
!********************************************************************
!
! Main Program for    spread_function
!
!********************************************************************
        PROGRAM Spread_Function
        USE LSQR_global
        IMPLICIT NONE
        EXTERNAL Aprod, LSQR
!
!
! Parameter specifications
```

```fortran
      !
            INTEGER, PARAMETER :: Leniw = 1, Lenrw = 1
            CHARACTER(*), PARAMETER :: Folder = "c:\Spread_function\"
      !
            character length 15
      !
      ! Other specifications
      !
      ! F is a vector representing the ground truth image stored by columns of
      ! length nf*nf.
      ! b is the filtered image stored by columns of length ng*ng.
      ! u, v , w, se are working arrays for LSQR
      ! h is a matrix representing the LADAR optics kernel or filter.
      !
            REAL, DIMENSION(:), ALLOCATABLE :: F, x, b, u, v, w, se, y, tmp, e, resid
            REAL, DIMENSION(:,:), ALLOCATABLE :: h
            DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: r1, r2
      !
            REAL, DIMENSION(Leniw) :: Iw
            REAL, DIMENSION(Lenrw) :: Rw
            REAL :: xx, atol, btol, conlim, damp, anorm, acond, rnorm, arnorm, xnorm,&
                        x1, x2
            REAL :: xmaxnorm, residnorm
            INTEGER :: i, j, n, m, Mode, Inputstatus, Memorystatus, nout, istop, mout(16),&
                        itnlim, ma2, cc, dd, ee, ff, k1, k2, k3, k, iter, maxiter
            CHARACTER(120) :: BeamMatrix, BeamVector,  GT, GTL, Input, InputL,&
                                        FilteredVect,  &
                                        Printfile, Reconstruct,
      ReconstructL,PredGroundTruth,&
                                        GroundTruthL, BarCodes
            CHARACTER(50) :: Beam, BeamL, Filtered, FilteredL, GroundTruth,&
                        Output, OutputL
            CHARACTER(4) :: CharMa, CharMaR
            CHARACTER(12) :: BeamFMT


      !
      ! Get the parameter input file.
      !
            Input = Folder//"LSQR_Spread_Input.txt"
            InputL = ADJUSTL(Input)
            OPEN(UNIT = 2, FILE = InputL, IOSTAT = Inputstatus)
            IF (Inputstatus > 0) STOP "*** Error on opening unit 2 Input file ***"
            READ(UNIT = 2, FMT = '(2I5)') ma, ng
            Print *,ma,ng
            READ(UNIT = 2, FMT = '(A4)') CharMa
            Print *,charma
```

```
            READ(UNIT = 2, FMT = '(I4)') maxiter
            Print *, maxiter
            READ(UNIT = 2, FMT = '(A50)') GroundTruth
            Print *, Beam
            READ(UNIT = 2, FMT = '(A50)') Filtered
            Print *, Filtered
            READ(UNIT = 2, FMT = '(A50)') Output
            Print *, Output
            READ(UNIT = 2, FMT = '(A50)') Beam
            Print *, Reconstruct
            READ(UNIT = 2, FMT = '(F5.3)') damp
            Print *, damp
            READ(UNIT = 2, FMT = '(F8.6)') atol
            Print *, atol
            READ(UNIT = 2, FMT = '(F8.6)') btol
            Print *, btol
            CLOSE(2)
            BeamL = ADJUSTL(Beam)
            BeamMatrix = Folder//'Mat_'//BeamL
            BeamVector = Folder//'Vect_'//BeamL
            FilteredL = ADJUSTL(Filtered)
            OutputL = ADJUSTL(Output)
            GroundTruthL = ADJUSTL(GroundTruth)
            FilteredVect = Folder//FilteredL
            Printfile = Folder//OutputL
            BarCodes = Folder//GroundTruthL
            CharMaR = ADJUSTR(CharMa)
            BeamFMT = '('//CharMaR//'G15.6)'
!
! Allocate arrays
!
! Get lengths of ground truth image vector, n, and filtered image vecctor, m.
!
            nf = ng + ma -1
            ma2 = ma*ma
            n = nf*nf
            m = ng*ng
            Print *, nf, ma2, n, m
            ALLOCATE(A(m,ma2), AT(ma2,m), x(ma2), b(m), u(m), v(ma2), w(ma2),&
                        se(ma2), F(n), h(ma,ma), y(m),tmp(m),e(ma2), r1(m), &
                        r2(m), resid(m), STAT = Memorystatus)
            IF (Memorystatus /= 0) STOP "*** Memory allocation error ***"
            Print *, 'Memory allocated'
!
! Read ground truth image
!
```

```fortran
        OPEN(UNIT = 8, FILE = BarCodes, STATUS = 'UNKNOWN')
        Print *, 'Barcode file = ', BarCodes
        i = 0
        DO
                READ (UNIT = 8, FMT = '(F8.4,F9.4,F7.2)', IOSTAT = Inputstatus)
x1,x2, xx

                IF (Inputstatus > 0) STOP "*** Input Error on Unit 8: Barcodes ***"
                IF (Inputstatus < 0) EXIT ! End of file
                i = i+1
                F(i) = xx
                END DO
        CLOSE(8)
        Print *, 'Barcodes', i, n


!
! Get the LADAR scanned or filtered image
!
        OPEN(UNIT = 6, FILE = FilteredVect,      STATUS = 'UNKNOWN')
        i = 0
        DO
!               READ(UNIT = 6,FMT = '(F13.6)',IOSTAT = Inputstatus)  xx
                READ (UNIT = 6, FMT = '(F8.4,F9.4,F7.2)', IOSTAT =&
                                Inputstatus)x1,x2, xx
                IF (Inputstatus > 0) STOP "*** Input Error on Unit 6 ***"
                IF (Inputstatus < 0) EXIT ! End of file
                i = i + 1
                b(i) = xx
        END DO
        CLOSE(6)
        Print *, 'Filtered image', i, m
!
! Convert Ground truth vector F to matrices A and AT
!
        DO cc = 0, ng-1
                DO dd = 0, ng-1
                        DO ee = 0, ma-1
                                DO ff = 1,ma
                                        k1 = cc*ng+dd+1
                                        k2 = ee*ma+ff
                                        k3 = (ee+cc)*nf + dd + ff
                                        A(k1,k2) = F(k3)
                                END DO
                        END DO
                END DO
        END DO
        DO cc = 1,m
```

```fortran
                DO dd = 1,ma2
                        AT(dd,cc) = A(cc,dd)
                END DO
        END DO
!
! Assign b to u for initial input to LSQR. Note u is overwritten.
!
        u = b
! Make b double precision
        r1 = b
!
! Dummy values. Work arrays not used.
!
        Iw(1) = 0
        Rw(1) = 0.0
!
! Setting up parameters for LSQR
!
!
        nout = 7
        OPEN(UNIT = nout, FILE = Printfile,        STATUS = 'UNKNOWN')
        conlim = 1.0e+7
        itnlim = 50
!
        iter = 0
!
! Begin the iterative refinement procedure to estimate the filter
!
! Determine initial kernel guess x on iteration 0
!
        Print *, 'Initial kernel and Residual Correction, Iteration = ', iter
        Call LSQR( m,ma2,Aprod,damp,Leniw,Lenrw,Iw,Rw,u,v,w,x,se,&
                atol,btol,conlim,itnlim,nout,istop,anorm,&
            acond,rnorm,arnorm,xnorm)
!
! Compute residual
!
        Mode = 1
        y = 0.0
        Call Aprod(Mode, m,ma2,x,y,Leniw,Lenrw, Iw,Rw)
        r2 = y   ! Make y = Ax double precision
        resid = r1 - r2 ! r = b - Ax
        tmp = resid
        residnorm = MAXNORM(m,tmp)
        Print *, 'Residual maxnorm, iteration 0', residnorm
!
```

```fortran
! pass in the initial residual to start the updates
!
        u = resid
!
        DO
        iter = iter + 1
!
! get the correction term and update the beam vector
! i.e. solve Ae = r
!
        Call LSQR( m,ma2,Aprod,damp,Leniw,Lenrw,Iw,Rw,u,v,w,e,se,&
                atol,btol,conlim,itnlim,nout,istop,anorm,&
              acond,rnorm,arnorm,xnorm)
!
! update kernel approximation
!
        x = x + e
!
! compute the new residual
!
        Mode = 1
        y = 0.0
        Call Aprod(Mode, m,ma2,x,y,Leniw,Lenrw, Iw,Rw)
        r2 = y   ! Make y = Ax double precision
        resid = r1 - r2 ! r = b - Ax
        tmp = resid
        residnorm = MAXNORM(m,tmp)
        Print *, 'Residual maxnorm, iteration =', iter, residnorm
!
! pass in the new residual to get the next update
!
        u = resid
!
! go back for the next correction
!
        IF (iter >= maxiter) EXIT
!
        END DO
        CLOSE(nout)
!
        x = x - e
!
! Write out the final filter
!
        OPEN(UNIT = 4, FILE = BeamVector, STATUS = 'UNKNOWN')
        DO i = ma2,1,-1
```

```fortran
               WRITE(UNIT = 4, FMT = '(G15.6)') x(i)
        END DO
        CLOSE(4)
        DO i = 1,ma
                DO j = 1,ma
                        k = (i-1)*ma + j
                        h(ma-(j-1),ma-(i-1)) = x(k)
                END DO
        END DO
        OPEN(UNIT = 4, FILE = BeamMatrix, STATUS = 'UNKNOWN')
        DO i = 1,ma
                WRITE(UNIT = 4, FMT = BeamFMT) (h(i,j), j = 1,ma)
        END DO
        CLOSE(4)
!
! Deallocate memory
!
        DEALLOCATE(A, AT, x, b, u, v, w, se, F )
!
CONTAINS
!********************************************************************
!
 FUNCTION MAXNORM(m, y)
!
!********************************************************************
!
! This function overwrites y
!
        IMPLICIT NONE
        REAL :: MAXNORM
        INTEGER :: m, i
        REAL, INTENT(INOUT) :: y(m)
        DO i = 1,m
                y(i) = ABS(y(i))
        END DO
        MAXNORM = MAXVAL(y)
!
END FUNCTION MAXNORM
!
!
END PROGRAM Spread_Function
!
!********************************************************************
!
        SUBROUTINE Aprod (Mode, m, n, x, y, Leniw, Lenrw, Iw, Rw)
!
```

```
!
! This version of Aprod performs a convolution of a filter matrix, h, with an image
vector, x.
! It also applies the transpose of h to a filtered image vector y. The storage
! of images is by columns.
!
! The beam vector is stored in x of length n. It is generated from a square image
! of side ma so that n = ma*ma. If h(i,j), i, j = 1,ma is the beam, the structure of x is as
follows:
! x(1) = h(ma,ma), x(2) = h(ma-1,ma), ..., x(ma) = h(1, ma), x(ma + 1) = x(ma, ma-1), ...,
x(2*ma) = h(1, ma-1),
! ..., x(ma*ma) = h(1, 1).
!
! The blurred image vector is stored in y of length m. It is generated from a square
! image of side ng so that m = ng*ng. If, g(i, j), i, j = 1, ng is the blurred image, the
! structure of y is as follows:
! y(1) = g(1, 1), y(2) = g(2, 1), ..., y(ng) = g(ng, 1), y(ng + 1) = g(1, 2), ..., y (2*ng) =
! g(ng, 2), ..., y(ng*ng) = g(ng, ng)
!
! The matrix A is the sparse block Toeplitz matrix formed from the ground truth image of
! size ng*ng X ma*ma. AT is the transpose
!
! Aprod performs the following function:
!
!       If Mode = 1, set y = y + A*x
!       If Mode = 2, set x = x + (A^T) * y
!
! Iw and Rw are work arrays of length Leniw and Lenrw respectively. They are not
! used in this version of Aprod
!
!********************************************************************************
!
        USE LSQR_global
!
! This subroutine needs to be placed in a module to access global values for
! h, ma, nf, ng defined below
!
        IMPLICIT NONE
        INTEGER :: n, m, Leniw, Lenrw, Mode
        INTEGER :: i, j, ng2, ma2
        REAL :: x(n), y(m), Iw(Leniw), Rw(Lenrw)
        REAL :: sum
!
!
        ng2 = ng*ng
        ma2 = ma*ma
```

```
!
!
!
!*********************************************************
!
!       Select the mode
!
!*********************************************************
!
        IF (mode == 1) THEN
!
!*********************************************************
!
!       Mode = 1  -- Set y = y + A*x
!
!*********************************************************
!
        DO i = 1,ng2
                sum = 0.0
                DO j = 1,ma2
                        sum = sum + A(i,j)*x(j)
                END DO
                y(i) = y(i) + sum
        END DO
!
        ELSE
!
!*********************************************************
!
!       Mode = 2 -- Set x = x + (A^T)*y
!
!*********************************************************
!
        DO i = 1,ma2
                sum = 0.0
                DO j = 1,ng2
                        sum = sum + AT(i,j)*y(j)
                END DO
                x(i) = x(i) + sum
        END DO
!
        END IF
!
        END SUBROUTINE Aprod
!
```

C5.3 Sample Input Parameter File

```
  11  095
  11
0100
yzint_Bar_1in_10m_data_105x105.txt
yzint_a_1in_10m_crop_95.txt
LSQR_output_1in_10m_95.txt
Beam_est_a_1in_10m_11x11.txt
0.001
0.0001
0.0001
```